Fabric Manager for NVIDIA NVSwitch Systems

User Guide / Virtualization / High Availability Modes

DU-09883-001_v0.7 | January 2021

Document History

DU-09883-001_v0.7

Version	Date	Authors	Description of Change
0.1	Oct 25, 2019	SB	Initial Beta Release
0.2	Mar 23, 2020	SB	Updated error handling and bare metal mode
0.3	May 11, 2020	YL	Updated Shared NVSwitch APIs section with new API information
0.4	July 7, 2020	SB	Updated MIG interoperability and high availability details.
0.5	July 17, 2020	SB	Updated running as non-root instructions
0.6	Aug 03, 2020	SB	Updated installation instructions based on CUDA repo and updated SXid error details
0.7	Jan 26, 2021	GT, CC	Updated with vGPU multitenancy virtualization mode

Table of Contents

Chapter 1. Overview

1.1 Introduction

NVIDIA DGX™ A100 and NVIDIA HGX™ A100 8-GPU [1] server systems use NVIDIA® NVLink® switches (NVIDIA® NVSwitch™) which enable all-to-all communication over the NVLink fabric. The DGX A100 and HGX A100 8-GPU systems both consist of a GPU baseboard, with eight NVIDIA A100 GPUs and six NVSwitches. Each A100 GPU has two NVLink connections to each NVSwitch on the same GPU baseboard. Additionally, two GPU baseboards can be connected together to build a 16-GPU system. Between the two GPU baseboards, the only NVLink connections are between NVSwitches with each switch from one GPU baseboard connected to a single NVSwitch on the other GPU baseboard for a total of sixteen NVLink connections

1.2 Terminology

Abbreviations	Definitions
FM	Fabric Manager
MMIO	Memory Mapped IO
VM	Virtual Machine
GPU register	A location in the GPU MMIO space
SBR	Secondary Bus Reset
DCGM	NVIDIA Data Center GPU manager
NVML	NVIDIA Management Library
Service VM	A privileged VM where NVIDIA NVSwitch software stack runs
Access NVLink	NVLink between a GPU and an NVSwitch
Trunk NVLink	NVLink between two GPU baseboards
SMBPBI	NVIDIA SMBus Post-Box Interface

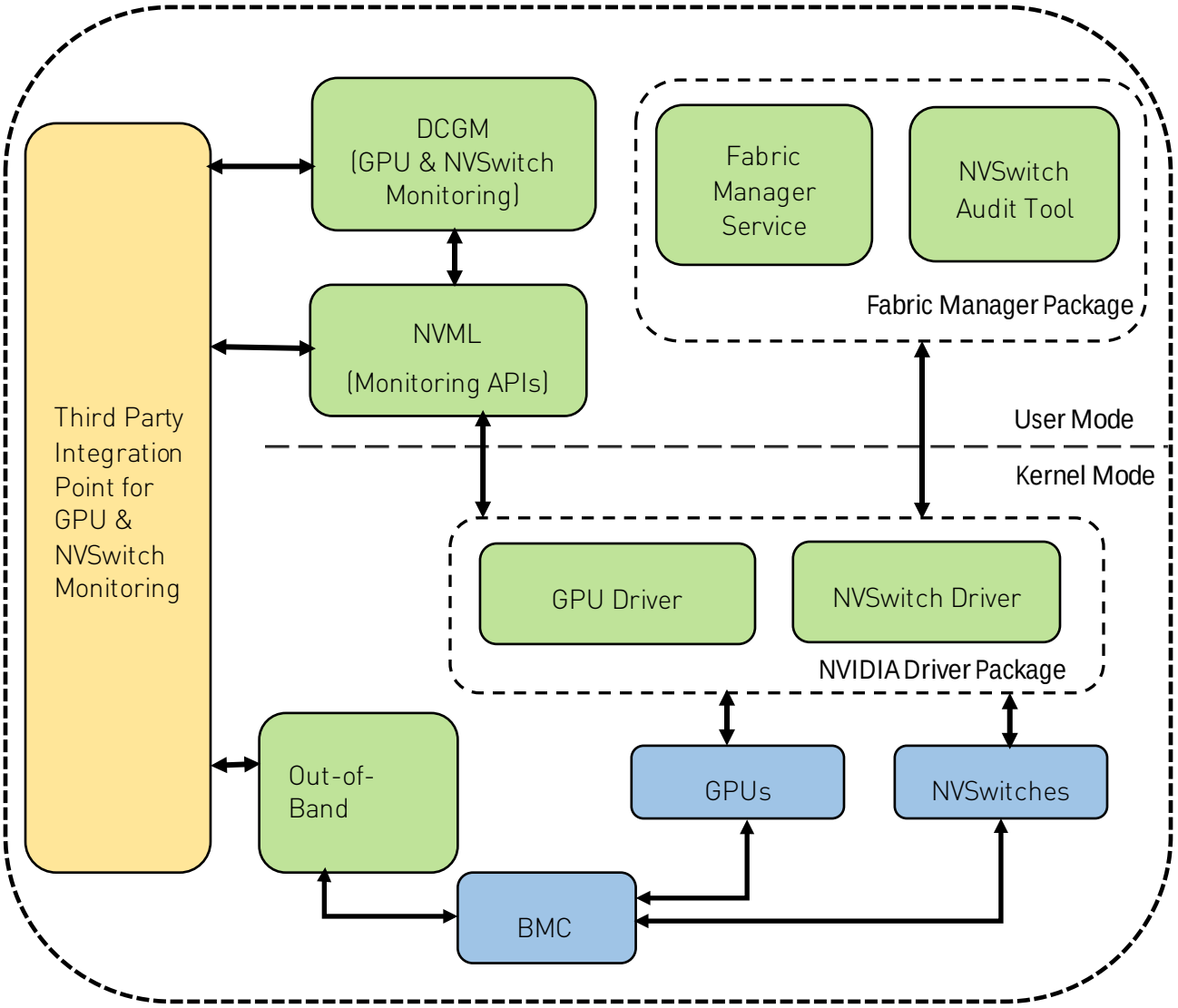
[1] The NVIDIA HGX A100 8-GPU will be referred to as the HGX A100 in the rest of this document.

Abbreviations	Definitions
vGPU	NVIDIA GRID Virtual GPU
MIG	Multi-Instance GPU
SR-IOV	Single-Root IO Virtualization
PF	Physical Function
VF	Virtual Function
GFID	GPU Function Identification
Partition	A collection of GPUs which are allowed to perform NVLink Peer-to-Peer Communication among themselves

1.3 NVSwitch Core Software Stack

The core software stack required for NVSwitch management consists of a NVSwitch kernel driver and a privileged process called Fabric Manager (FM). The kernel driver performs low level hardware management in response to Fabric Manager requests. The software stack also provides in-band and out-of-band monitoring solutions for reporting both NVSwitch and GPU errors and status information.

Figure 1. NVSwitch core software stack



1.4 What is Fabric Manager?

NVIDIA Fabric Manager (FM) configures the NVSwitch memory fabrics to form a single memory fabric among all participating GPUs and monitors the NVLinks that support the fabric. At a high level, Fabric Manager has the following responsibilities

- Coordinate with NVSwitch driver to initialize and train NVSwitch to NVSwitch NVLink interconnects.
- Coordinate with GPU driver to initialize and train NVSwitch to GPU NVLink interconnects.
- Configure routing among NVSwitch ports.

- Monitor the fabric for NVLink and NVSwitch errors.

This document provides an overview of various Fabric Manager features and is intended for system administrators and individual users of NVSwitch-based server systems.

Chapter 2. Getting Started With Fabric Manager

2.1 Basic Components

2.1.1 Fabric Manager Service

The core component of Fabric Manager is implemented as a standalone executable which runs as a Unix Daemon process. The FM installation package will install all the required core components and register the daemon as a system service named *nvidia-fabricmanager*.

2.1.2 Software Development Kit

Fabric Manager also provides a shared library, a set of C/C++ APIs (SDK) and corresponding header files. These APIs are used to interface with Fabric Manager service to query/activate/deactivate GPU partitions when Fabric Manager is running in Shared NVSwitch and vGPU multi-tenancy modes. All these SDK components are installed through a separate development package. For more information, refer to the Shared NVSwitch and vGPU Virtualization Model chapters.

2.2 NVSwitch and NVLink Initialization

NVIDIA GPUs and NVSwitch memory fabrics are PCIe endpoint devices requiring an NVIDIA kernel driver to be used. After system boot, none of the NVLink connections are enabled until the NVIDIA kernel driver is loaded, and the Fabric Manager configures them. In an NVSwitch-based system, CUDA initialization will fail with error *cudaErrorSystemNotReady* if the application is launched either before Fabric Manager fully initializes the system or when Fabric Manager fails to initialize the system.

) **Note:** System administrators can set their GPU application launcher services (such as SSHD, Docker etc.) to start after the Fabric Manager service is started. Refer to your Linux distribution's manual for setting up service dependency and service start order for the same.

2.3 Supported Platforms

Fabric Manager currently supports the following products and environments:

2.3.1 Hardware Architectures

- x86_64

2.3.2 NVIDIA Server Architectures

- DGX A100 and HGX A100 systems using A100 GPUs and compatible NVSwitch

2.3.3 OS Environment

Fabric Manager is supported on the following major Linux OS distributions.

- RHEL/CentOS 7.x and RHEL/CentOS 8.x
- Ubuntu 16.04.x, Ubuntu 18.04.x, and Ubuntu 20.04.x

2.4 Supported Deployment Models

NVSwitch-based systems can be deployed as bare metal servers or in a virtualized (full passthrough, shared NVSwitch, or vGPU) multi-tenant environment. Fabric Manager supports all these deployment models and refer to their respective chapters for more information.

2.5 Other NVIDIA Software Packages

To run Fabric Manager service, the target system must include the following NVIDIA components

- Compatible Driver for NVIDIA Data Center GPUs (Starting with version R450)

> **Note:** During initialization time, Fabric Manager service checks the currently loaded kernel driver stack version for compatibility and aborts if the loaded driver stack version is not compatible.

2.6 Installation

2.6.1 On DGX A100 Systems

Fabric Manager service is pre-installed in all the NVSwitch-based DGX A100 systems as part of the supported DGX OS package. The service is enabled and started on OS boot.

2.6.2 On HGX A100 Systems

On NVSwitch-based HGX A100 systems, the Fabric Manager service needs to be manually installed in order to configure the NVLinks and NVSwitch memory fabrics to support a single memory fabric. The Fabric Manager package is available through the NVIDIA CUDA network repository. Follow the instructions at https://docs.nvidia.com/datacenter/tesla/tesla-installation-notes/index.html to set up your system's package manager to download packages from the desired CUDA network repositories.

Once the package manager network repositories are set up, use the following distro-specific Fabric Manager installation command. Note that <driver-branch> should be substituted with the required NVIDIA driver branch number for qualified datacenter drivers (for example, 450) in the instructions below.

- For Debian and Ubuntu based OS distributions
```
sudo apt-get install cuda-drivers-fabricmanager-<driver-branch>
```
- For Red Hat Enterprise Linux 8 based OS distributions
```
sudo dnf module install nvidia-driver:<driver-branch>/fm
```
- SUSE Linux based OS distributions
```
sudo zypper install cuda-drivers-fabricmanager-<driver-branch>
```

Note: On NVSwitch-based NVIDIA HGX A100 systems, install the compatible Driver for NVIDIA Data Center GPUs before installing Fabric Manager. Also as part of installation, the FM service unit file (nvidia-fabricmanager.service) will be copied to systemd location. However, the system administrator must manually enable and start the Fabric Manager service.

2.7 Managing Fabric Manager Service

2.7.1 Start Fabric Manager

```
# For Linux based OS distributions
sudo systemctl start nvidia-fabricmanager
```

2.7.2 Stop Fabric Manager

```
# For Linux based OS distributions
```

```
sudo systemctl stop nvidia-fabricmanager
```

2.7.3 Check Fabric Manager Status

```
# For Linux based OS distributions
sudo systemctl status nvidia-fabricmanager
```

2.7.4 Enable FM Service to Auto Start at Boot

```
# For Linux based OS distributions
sudo systemctl enable nvidia-fabricmanager
```

2.7.5 Disable FM Service Auto Start at Boot

```
# For Linux based OS distributions
sudo systemctl disable nvidia-fabricmanager
```

2.7.6 Check Fabric Manager System Log Messages

```
# For Linux based OS distributions
sudo journalctl -u nvidia-fabricmanager
```

2.8 Fabric Manager Startup Options

Fabric Manager supports the following command line options

```
. /nv-fabricmanager -h

    NVIDIA Fabric Manager
    Runs as a background process to configure the NVSwitches to form
    a single memory fabric among all participating GPUs.

    Usage: nv-fabricmanager [options]

    Options include:
    [-h | --help]:              Displays help information
    [-v | --version]:           Displays the Fabric Manager version and exit.
    [-c | --config]:            Provides Fabric Manager config file path/name
which controls all the config options.
    [-r | --restart]:           Restart Fabric Manager after exit. Applicable
to Shared NVSwitch and vGPU multitenancy modes.
```

Most of the Fabric Manager configurable parameters and options are specified through a text config file. Fabric Manager installation will copy a default config file to a predefined location and the same will be used by default. To use a different config file location, specify the same using the *[-c / --config]* command line argument.

) **Note:** On Linux based installations, the default Fabric Manager config file will be at /usr/share/nvidia/nvswitch/fabricmanager.cfg. Also, if the default config file on the

system is modified, any subsequent Fabric Manager package update will provide options to manage (such as merge/keep/overwrite) the existing config file.

2.9 Fabric Manager Service File

2.9.1 On Linux Based Systems

On Linux based systems, the installation package will register Fabric Manager service using the following systemd service unit file. Modify this service unit file located under */lib/systemd/system/nvidia-fabricmanager.service* to change Fabric Manager service start-up options.

```
[Unit]
Description=FabricManager service
After=network-online.target
Requires=network-online.target

[Service]
User=root
PrivateTmp=false
Type=forking

ExecStart=/usr/bin/nv-fabricmanager -c /usr/share/nvidia/nvswitch/fabricmanager.cfg

[Install]
WantedBy=multi-user.target
```

2.10 Running Fabric Manager as Non-Root

On Linux based systems, by default Fabric Manager service requires administrative (root) privileges to configure all the GPU NVLinks and NVSwitches to support a single memory fabric. However, system administrators and advanced users can follow the below steps to run Fabric Manager from a non-root account.

- Stop the Fabric Manager instance if it is already running.
- Fabric Manager requires access to the following directory/file. Adjust corresponding directory/file access to the desired user/user group.

  `/var/run/nvidia-fabricmanager` – Fixed location to save runtime information.

  `/var/log/` - Configurable location to save Fabric Manager log file.

  `/usr/share/nvidia/nvswitch` – Configurable location for fabric topology files.

  The above configurable directory/file information is based on default Fabric Manager config file options. If the default configuration values are changed, adjust the directory/file information accordingly.

- NVIDIA driver will create the following proc entry with default permission to root. Change its read/write access to the desired user/user group
  `/proc/driver/nvidia-nvlink/capabilities/fabric-mgmt`
- Fabric Manager also requires access to the following device node files. By default, these device node files are created by *nvidia-modprobe* utility (installed as part of NVIDIA Driver package for Data Center GPUs) with access permission for all users. If these device node files are created manually or outside of *nvidia-modprobe*, then assign read/write access to the desired user/user group.
  `/dev/nvidiactl`
  `/dev/nvidia-nvlink`
  `/dev/nvidia-nvswitchctl`
  `/dev/nvidiaX` (one for each GPU device)
  `/dev/nvidia-nvswitchX` (one for each NVSwitch device)
- Once all the above required permissions are assigned, start the Fabric Manager process from the desired user/user group account manually.
- The NVIDIA driver will create/re-create the above proc entry during driver load. So, the above steps need to be repeated on every driver reload or system boot.

When Fabric Manager is configured as a systemd service, system administrator must edit the Fabric Manager service unit file to instruct systemd to run Fabric Manager service from a specific user/group. This specific user/group can be specified through the *User=* and *Group=* directive in the *[Service]* section of Fabric Manager service unit file. System administrator must ensure that the above proc entry and associated file node permission are changed to desired user/user group before Fabric Manager service starts at system boot time.

Also, when Fabric Manager is configured to run from a specific user/user group as specified above, the *nvswitch-audit* command line utility should be started from the same user/user group account.

Note: System administrators can set up necessary udev rules to automate the process of changing the above proc entry permissions.

2.11 Fabric Manager Config Options

All the configurable parameters and options used by Fabric Manager are specified through a text config file. The following section lists all those currently supported configurable parameters and options.

Note: The Fabric Manager config file is read as part of Fabric Manager service startup. So, after changing any config file options, Fabric Manager service needs to be restarted for the new settings to take effect.

2.11.1 Logging Related Config Items

2.11.1.1 Setting Log File Location and Name

Config Item:

LOG_FILE_NAME=<value>

Supported/Possible Values:

Full path/filename string (max length of 256) for the log.

Default Value:

LOG_FILE_NAME=/var/log/fabricmanager.log

2.11.1.2 Setting Desired Log Level

Config Item:

LOG_LEVEL=<value>

Supported/Possible Values:

0 – All the logging is disabled
1 - Set log level to CRITICAL and above
2 - Set log level to ERROR and above
3 - Set log level to WARNING and above
4 - Set log level to INFO and above

Default Value:

LOG_LEVEL=4

2.11.1.3 Setting Log File Append Behavior

Config Item:

LOG_APPEND_TO_LOG=<value>

Supported/Possible Values:

0 – No, don't append to the existing log file, instead overwrite the existing log file.

1 – Yes, append to the existing log file every time Fabric Manager service is started.

Default Value:

LOG_APPEND_TO_LOG=1

2.11.1.4 Setting Log File Size

Config Item:

LOG_FILE_MAX_SIZE=<value>

Supported/Possible Values:

The desired max log file size in MBs. Once the specified size is reached, Fabric Manager will skip further logging to the specified log file.

Default Value:

LOG_FILE_MAX_SIZE=1024

2.11.1.5 Redirect Logs to Syslog

Config Item:

LOG_USE_SYSLOG=<value>

Supported/Possible Values:

0 – Use the specified log file for storing all the Fabric Manager logs
1 - Redirect all the Fabric Manager logs to syslog instead file based logging.

Default Value:

LOG_USE_SYSLOG=0

Note: Fabric Manager log is in clear text format and NVIDIA recommends running Fabric Manager service with logging enabled at INFO level for troubleshooting field issues.

2.11.2 Operating Mode Related Config Items

Note: This section of config items is applicable only to Shared NVSwitch and vGPU Multitenancy deployment.

2.11.2.1 Fabric Manager Operating Mode

Config Item:

FABRIC_MODE=<value>

Supported/Possible Values:

0 – Start Fabric Manager in bare metal or full pass through virtualization mode

1 - Start Fabric Manager in Shared NVSwitch multitenancy mode. For more information, refer to the Shared NVSwitch Virtualization Model chapter.

2 - Start Fabric Manager in vGPU multitenancy mode. For more information, refer to the vGPU Virtualization Model chapter.

Default Value:

```
FABRIC_MODE=0
```

Note: Older SHARED_FABRIC_MODE configuration item is still supported but it is recommended to use FABRIC_MODE configuration item.

2.11.2.2 Fabric Manager Restart Mode

Config Item:

```
FABRIC_MODE_RESTART=<value>
```

Supported/Possible Values:

0 - Start Fabric Manager and follow full initialization sequence

1 - Start Fabric Manager and follow Shared NVSwitch or vGPU multitenancy mode resiliency/restart sequence.

This option is equivalent to the *–restart* command line argument to the Fabric Manager process and is provided to enable the Shared NVSwitch or vGPU multitenancy mode resiliency without modifying command line arguments to Fabric Manager process. For more information on the FM resiliency flow, refer to *Appendix C. Resiliency*.

Default Value:

```
FABRIC_MODE_RESTART=0
```

Note: Older SHARED_FABRIC_MODE_RESTART configuration item is still supported but it is recommended to use FABRIC_MODE_RESTART configuration item.

2.11.2.3 Fabric Manager API Interface

Config Item:

```
FM_CMD_BIND_INTERFACE =<value>
```

Supported/Possible Values:

Network interface for Fabric Manager SDK/API to listen, for Hypervisor to communicate with running FM instance for Shared NVSwitch and vGPU multitenancy operations.

Default Value:

FM_CMD_BIND_INTERFACE=127.0.0.1

2.11.2.4 Fabric Manager API TCP Port

Config Item:

FM_CMD_PORT_NUMBER=<value>

Supported/Possible Values:

TCP port number for Fabric Manager SDK/API for Hypervisor to communicate with running FM instance for Shared NVSwitch and vGPU multitenancy operations.

Default Value:

FM_CMD_PORT_NUMBER=6666

2.11.2.5 Fabric Manager Domain Socket Interface

Config Item:

FM_CMD_UNIX_SOCKET_PATH=<value>

Supported/Possible Values:

Unix domain socket path instead of TCP/IP socket for Fabric Manager SDK/API to listen, to communicate with running FM instance for Shared NVSwitch and vGPU multitenancy operations.

Default Value:

FM_CMD_UNIX_SOCKET_PATH=<empty value>

2.11.2.6 Fabric Manager State

Config Item:

STATE_FILE_NAME=<value>

Supported/Possible Values:

Specify the filename to be used to save Fabric Manager states for restarting Fabric Manager after crash or successful exit. Valid only if Shared NVSwitch or vGPU multitenancy mode is enabled.

Default Value:

STATE_FILE_NAME =/tmp/fabricmanager.state

2.11.3 Miscellaneous Config Items

2.11.3.1 Prevent Fabric Manager from daemonizing

Config Item:

DAEMONIZE=<value>

Supported/Possible Values:

0 – Do not daemonize and run fabric manager as a normal process
1 – Run Fabric Manager process as a Unix daemon.

Default Value:

DAEMONIZE=1

2.11.3.2 Fabric Manager Communication Socket Interface

Config Item:

BIND_INTERFACE_IP=<value>

Supported/Possible Values:

Network interface to listen for Fabric Manager internal communication/IPC. Should be a valid IPv4 address.

Default Value:

BIND_INTERFACE_IP=127.0.0.1

2.11.3.3 Fabric Manager Communication TCP Port

Config Item:

STARTING_TCP_PORT=<value>

Supported/Possible Values:

Starting TCP port number for Fabric Manager internal communication/IPC. Should be a value between 0 and 65535.

Default Value:

STARTING_TCP_PORT=16000

2.11.3.4 Unix Domain Socket for Fabric Manager Communication

Config Item:

UNIX_SOCKET_PATH=<value>

Supported/Possible Values:

Use Unix Domain socket instead of TCP/IP socket for Fabric Manager internal communication/IPC. Empty value means Unix Domain socket is not used.

Default Value:

UNIX_SOCKET_PATH=<empty value>

2.11.3.5 Fabric Manager System Topology File Location

Config Item:

TOPOLOGY_FILE_PATH =<value>

Supported/Possible Values:

Configuration option to specify the Fabric Manager topology files directory path information.

Default Value:

TOPOLOGY_FILE_PATH=/usr/share/nvidia/nvswitch

2.11.4 High Availability Mode related config items

2.11.4.1 Control Fabric Manager behavior on initialization failure

Config Item:

FM_STAY_RESIDENT_ON_FAILURES=<value>

Supported/Possible Values:

0 – Fabric Manager service will terminate on errors such as, NVSwitch and GPU config failure, typical software errors etc.

1 – Fabric Manager service will stay running on errors such as, NVSwitch and GPU config failure, typical software errors etc. However, the system will be uninitialized and CUDA application launch will fail.

Default Value:

FM_STAY_RESIDENT_ON_FAILURES=0

2.11.4.2 GPU Access NVLink Failure Mode

Config Item:

ACCESS_LINK_FAILURE_MODE=<value>

Supported/Possible Values:

Available high availability options when there is an Access NVLink Failure (GPU to NVSwitch NVLink failure). For more information about supported values and behavior, refer to Supported High Availability Modes Chapter.

Default Value:

ACCESS_LINK_FAILURE_MODE=0

2.11.4.3 NVSwitch Trunk NVLink Failure Mode

Config Item:

TRUNK_LINK_FAILURE_MODE=<value>

Supported/Possible Values:

Available high availability options when there is a Trunk Link failure (NVSwitch to NVSwitch connection between GPU baseboards). For more information about supported values and behavior, refer to Supported High Availability Modes Chapter.

Default Value:

TRUNK_LINK_FAILURE_MODE=0

2.11.4.4 NVSwitch Failure Mode

Config Item:

NVSWITCH_FAILURE_MODE=<value>

Supported/Possible Values:

Available high availability options when there is an NVSwitch failure. For more information about supported values and behavior, refer to Supported High Availability Modes Chapter.

Default Value:

NVSWITCH_FAILURE_MODE=0

2.11.4.5 CUDA Jobs Behavior when Fabric Manager Service is Stopped or Terminated

Config Item:

ABORT_CUDA_JOBS_ON_FM_EXIT=<value>

Supported/Possible Values:

0 – Do not abort running CUDA jobs when Fabric Manager service is stopped or exits. However, a new CUDA job launch will fail with `cudaErrorSystemNotReady` error.
1 – Abort all running CUDA jobs when Fabric Manager service is stopped or exits. Also, a new CUDA job launch will fail with `cudaErrorSystemNotReady` error.

This config option is applicable to only bare metal and full pass through virtualization models.

Default Value:

```
ABORT_CUDA_JOBS_ON_FM_EXIT=1
```

Chapter 3. Bare Metal Mode

3.1 Introduction

The NVIDIA DGX A100 and HGX A100 systems' default software configuration is to run them as bare-metal machines for all workloads (such as AI, machine learning, etc.). The following section details Fabric Manager installation requirements for supporting bare metal configuration.

3.2 Fabric Manager Installation

3.2.1 On DGX A100 Systems

Fabric Manager service is pre-installed in all the NVSwitch-based DGX A100 systems as part of the supported DGX OS package installation. The service is enabled and started on OS boot and the default installation configuration is to support bare metal mode.

3.2.2 On HGX A100 Systems

In order to configure NVSwitch-based NVIDIA HGX A100 systems for bare metal mode, system administrators must install the following software packages.

- Driver package for NVIDIA Data Center GPUs (Version 450.xx and higher)
- NVIDIA Fabric Manager Package (Same version as Driver package)

Fabric Manager default installation mode and configuration file options are for supporting bare metal mode.

3.3 Runtime NVSwitch and GPU errors

When an NVSwitch port or GPU generates runtime error, the corresponding information will be logged into the operating system's kernel log or event log. Fabric Manager will also log the same to its log file and syslog. Error report from NVSwitch will be logged

with a prefix *SXid* and GPU error report will be logged with a prefix *Xid* by the NVIDIA driver.

All the NVSwitch *SXids* errors use the following reporting convention,

```
<nvidia-nvswitchX: SXid (PCI:<switch_pci_bdf>):  <SXid_Value>, <Fatal
or Non-Fatal>, <Link No> < Error Description>
<raw error information for additional troubleshooting>

The following is an example of a SXid error log
[...] nvidia-nvswitch2: SXid (PCI:0000:15:00.0): 20034, Fatal, Link 28
LTSSM Fault Up
[...] nvidia-nvswitch2: SXid (PCI:0000:15:00.0): 20034, Severity 5
Engine instance 28 Sub-engine instance 00
[...] nvidia-nvswitch2: SXid (PCI:0000:15:00.0): 20034, Data
{0x50610002, 0x10100030, 0x00000000, 0x10100030, 0x00000000,
0x00000000, 0x00000000, 0x00000000}
```

All the GPU *Xids* errors use the following reporting convention,

```
NVRM: GPU at PCI:<gpu_pci_bdf>: <gpu_uuid>
NVRM: GPU Board Serial Number: <gpu_serial_number>
NVRM: Xid (PCI:<gpu_pci_bdf>): <Xid_Value>, <raw error information>

The following is an example of a Xid error log
[...] NVRM: GPU at PCI:0000:34:00: GPU-c43f0536-e751-7211-d7a7-
78c95249ee7d
[...] NVRM: GPU Board Serial Number: 0323618040756
[...] NVRM: Xid (PCI:0000:34:00): 45, Ch 00000010
```

Depending on the severity (Fatal vs Non-Fatal) and the impacted port, SXid and Xid errors can abort existing CUDA jobs and prevent new CUDA job launches. The following section details potential impact of SXid and Xid errors and corresponding recovery procedure.

3.3.1 NVSwitch SXid Errors

3.3.1.1 NVSwitch Non-Fatal SXid Errors

NVSwitch non-fatal SXids are mainly for informational purpose only and Fabric Manager do not terminate any running CUDA jobs or prevent new CUDA job launches. In general, existing CUDA jobs should resume; however, depending on the exact error, CUDA jobs may experience issues like performance drop, no forward progress for brief time etc.

3.3.1.2 NVSwitch Fatal SXid Errors

When a fatal SXid error is reported on a NVSwitch port, which is connected between a GPU and a NVSwitch, the corresponding error will be propagated to the GPU side. The running CUDA jobs on that GPU will be aborted and the GPU may report Xid 74 and Xid 45

errors. Fabric Manager service will log the corresponding GPU index and PCI bus information in its log file and syslog. The system administrator must use the following recovery procedure to clear the error state before using the GPU for further CUDA workload.

1. Reset the specified GPU (and all the participating GPUs in the affected workload) via the NVIDIA System Management Interface (nvidia-smi) command line utility.

   Refer to -r, or --gpu-reset options in nvidia-smi for more information and individual GPU reset operation.
2. If the problem persists, reboot or power cycle the system.

When a fatal SXid error is reported on a NVSwitch port, which connects two GPU baseboards, Fabric Manager will abort all the running CUDA jobs and prevent any new CUDA job launch. Also, the GPU will report Xid 45 error as part of aborting CUDA jobs. Fabric Manager service will log the corresponding error information in its log file and syslog. The system administrator must use the following recovery procedure to clear the error state and subsequent successful CUDA job launch.

1. Reset all the GPUs and NVSwitches
   a). Stop Fabric Manager service
   b). Stop/Kill all the applications which are using the GPU
   c). Reset all the GPU and NVSwitches via the NVIDIA System Management Interface (nvidia-smi) command line utility with only "-r" or "--gpu-reset" option. (i.e. don't use the "-i, or –id" option)
   d). Once the reset operation is done, start Fabric Manager service again
2. If the problem persists, reboot or power cycle the system

3.3.2 GPU Xid Errors

GPU Xid messages indicate that a general GPU error occurred. The messages can be indicative of a hardware problem, an NVIDIA software problem, or a user application problem. When a GPU experiences Xid error, most likely the running CUDA jobs on that GPU will be aborted. Follow the above-mentioned individual GPU reset procedure for further troubleshooting.

3.4 Interoperability With MIG

MIG (Multi-Instance GPUs) feature partitions a single NVIDIA A100 GPU into as many as seven independent GPU instances. They run simultaneously, each with its own memory, cache, and streaming multiprocessors. However, enabling the MIG mode will disable the GPU NVLinks and the GPU will lose its NVLink P2P (Peer-to-Peer) capability. Once the

MIG mode is successfully disabled, the GPU NVLinks will be re-enabled, and the GPU NVLink P2P capability will be restored. On DGX A100 and HGX A100 systems, Fabric Manager service can cooperate with GPU MIG instances. Also, Fabric Manager service is required to be running in order to successfully enable and disable GPU MIG mode at run-time without additional GPU reset operation and to restore the GPU NVLink P2P capability when MIG mode is disabled.

Chapter 4. Virtualization Models

4.1 Introduction

NVIDIA® NVSwitch™ based systems support multiple models for isolating NVLink interconnects in a multi-tenant environment. In virtualized environments, VM's workloads are often untrusted and must be isolated from each other and from the host or hypervisor. The switches used to maintain this isolation cannot be directly controlled by untrusted VMs and must instead be controlled by the trusted software.

This chapter provides a high-level overview of supported virtualization models.

4.2 Supported Virtualization Models

The NVSwitch-based systems support three different virtualization models that are described detail in subsequent chapters. Briefly,

- Full Passthrough
  - Both GPUs and NVSwitch memory fabrics are passed through to the Guest OS.
  - Easy to deploy requiring minimal changes to the Hypervisor/Host OS
  - Reduced bandwidth for 2 and 4 GPU VMs
  - Total 16 GPUs + 12 switches need to be passed through for the largest VM
- Shared NVSwitch Multitenancy Mode
  - Only GPUs passed through to the guests
  - NVSwitch memory fabrics are managed by a dedicated trusted VM called Service VM
  - NVSwitch memory fabrics are shared by all the Guest VMs, but are not visible to guests.
  - Requires the tightest integration with the Hypervisor
  - Full bandwidth for 2 and 4 GPU VMs
  - No need for any communication between Guest VM and Service VM
- vGPU Multitenancy Mode
  - Only SR-IOV GPU VFs passed through to the guests
  - GPU PFs and NVSwitch memory fabrics are managed by the vGPU host

- NVSwitch memory fabrics are shared by all the Guest VMs, but are not visible to guests.
- Full bandwidth for 2 and 4 GPU VMs
- This mode is tightly coupled with vGPU software stack

Chapter 5. Fabric Manager SDK

Fabric Manager provides a shared library, a set of C/C++ APIs (SDK) and corresponding header files. This shared library and corresponding APIs are used to interface with Fabric Manager when it is running in Shared NVSwitch and vGPU multi-tenant modes to query/activate/deactivate GPU partitions.

All Fabric Manager interface API definitions, libraries, sample code and associated data structure definitions are delivered as a separate development package (RPM/Debian). This package must be installed to compile the sample code provided in this user guide.

5.1 Data Structures

```
// max number of GPU/fabric partitions supported by FM
#define FM_MAX_FABRIC_PARTITIONS 64

// max number of GPUs supported by FM
#define FM_MAX_NUM_GPUS     16

// Max number of ports per NVLink device supported by FM
#define FM_MAX_NUM_NVLINK_PORTS  64

// connection options for fmConnect()
typedef struct
{
    unsigned int version;
    char addressInfo[FM_MAX_STR_LENGTH];
    unsigned int timeoutMs;
    unsigned int addressIsUnixSocket;
} fmConnectParams_v1;

typedef fmConnectParams_v1 fmConnectParams_t;

// VF PCI Device Information
typedef struct
{
    unsigned int domain;
    unsigned int bus;
    unsigned int device;
    unsigned int function;
} fmPciDevice_t;
```

```
// structure to store information about a GPU belonging to fabric partition
typedef struct
{
    unsigned int physicalId;
    char uuid[FM_UUID_BUFFER_SIZE];
    char pciBusId[FM_DEVICE_PCI_BUS_ID_BUFFER_SIZE];
    unsigned int numNvLinksAvailable;
    unsigned int maxNumNvLinks;
    unsigned int nvlinkLineRateMBps;
} fmFabricPartitionGpuInfo_t;

// structure to store information about a fabric partition
typedef struct
{
    fmFabricPartitionId_t partitionId;
    unsigned int isActive;
    unsigned int numGpus;
    fmFabricPartitionGpuInfo_t gpuInfo[FM_MAX_NUM_GPUS];
} fmFabricPartitionInfo_t;

// structure to store information about all the supported fabric partitions
typedef struct
{
    unsigned int version;
    unsigned int numPartitions;
    unsigned int maxNumPartitions;
    fmFabricPartitionInfo_t partitionInfo[FM_MAX_FABRIC_PARTITIONS];
} fmFabricPartitionList_v2;

typedef fmFabricPartitionList_v2 fmFabricPartitionList_t;

// structure to store information about all the activated fabric partitionIds
typedef struct
{
    unsigned int version;
    unsigned int numPartitions;
    fmFabricPartitionId_t partitionIds[FM_MAX_FABRIC_PARTITIONS];
} fmActivatedFabricPartitionList_v1;

typedef fmActivatedFabricPartitionList_v1 fmActivatedFabricPartitionList_t;

// Structure to store information about a NVSwitch or GPU with failed NVLinks
typedef struct
{
    char         uuid[FM_UUID_BUFFER_SIZE];
```

```
    char         pciBusId[FM_DEVICE_PCI_BUS_ID_BUFFER_SIZE];
    unsigned int numPorts;
    unsigned int portNum[FM_MAX_NUM_NVLINK_PORTS];
} fmNvlinkFailedDeviceInfo_t;

// Structure to store a list of NVSwitches and GPUs with failed NVLinks
typedef struct
{
    unsigned int               version;
    unsigned int               numGpus;
    unsigned int               numSwitches;
    fmNvlinkFailedDeviceInfo_t  gpuInfo[FM_MAX_NUM_GPUS];
    fmNvlinkFailedDeviceInfo_t  switchInfo[FM_MAX_NUM_NVSWITCHES];
} fmNvlinkFailedDevices_v1;

typedef fmNvlinkFailedDevices_v1 fmNvlinkFailedDevices_t;

/**
 * Structure to store information about a unsupported fabric partition
 */
typedef struct
{
    fmFabricPartitionId_t partitionId; //!< a unique id assigned to reference
this partition
    unsigned int numGpus;     //!< number of GPUs in this partition
    unsigned int gpuPhysicalIds[FM_MAX_NUM_GPUS];    //!< physicalId of each
GPU assigned to this partition.
} fmUnsupportedFabricPartitionInfo_t;
/**
 * Structure to store information about all the unsupported fabric partitions
 */
typedef struct
{
    unsigned int version;        //!< version number. Use
fmFabricPartitionList_version
    unsigned int numPartitions;   //!< total number of unsupported partitions
    fmUnsupportedFabricPartitionInfo_t partitionInfo[FM_MAX_FABRIC_PARTITIONS];
/*!< detailed information of each

unsupported partition*/
} fmUnsupportedFabricPartitionList_v1;
typedef fmUnsupportedFabricPartitionList_v1 fmUnsupportedFabricPartitionList_t;
#define fmUnsupportedFabricPartitionList_version1
MAKE_FM_PARAM_VERSION(fmUnsupportedFabricPartitionList_v1, 1)
#define fmUnsupportedFabricPartitionList_version
fmUnsupportedFabricPartitionList_version1
```

5.2 Initializing FM API interface

The Fabric Manager API interface library must be initialized first. Use the following method for the same.

```
fmReturn_t fmLibInit(void)

Parameters
       None
Return Values
       FM_ST_SUCCESS - if FM API interface library has been properly initialized
       FM_ST_IN_USE - FM API interface library is already in initialized state.
       FM_ST_GENERIC_ERROR - A generic, unspecified error occurred
```

5.3 Shutting Down FM API interface

This method is used to shut down the Fabric Manager API interface library. Any remote connections established through fmConnect() will be shut down as well.

```
fmReturn_t fmLibShutdown(void)

Parameters
       None
Return Values
       FM_ST_SUCCESS - if FM API interface library has been properly shut down
       FM_ST_UNINITIALIZED - interface library was not in initialized state.
```

5.4 Connect to Running FM Instance

The following method is used to connect to a running instance of Fabric Manager. Fabric Manager instance is started as part of system service or manually by the SysAdmin. This connection will be used by the APIs to exchange information to the running Fabric Manager instance.

```
fmReturn_t fmConnect(fmConnectParams_t *connectParams, fmHandle_t *pFmHandle)

Parameters
connectParams
       Valid IP address for the remote host engine to connect to. If ipAddress
       is specified as x.x.x.x it will attempt to connect to the default port
       specified by FM_CMD_PORT_NUMBER.If ipAddress is specified as x.x.x.x:yyyy
       it will attempt to connect to the port specified by yyyy. To connect to
       an FM instance that was started with unix domain socket fill the socket
       path in addressInfo member and set addressIsUnixSocket flag.
pfmHandle
```

```
      Fabric Manager API interface abstracted handle for subsequent API calls
Return Values
      FM_ST_SUCCESS - successfully connected to the FM instance
      FM_ST_CONNECTION_NOT_VALID - if the FM instance could not be reached
      FM_ST_UNINITIALIZED - FM interface library has not been initialized
      FM_ST_BADPARAM - pFmHandle is NULL or IP Address/format is invalid
      FM_ST_VERSION_MISMATCH - provided versions of params do not match
```

5.5 Disconnect from Running FM Instance

The following method is used to disconnect from a Fabric Manager instance.

```
fmReturn_t fmDisconnect(fmHandle_t pFmHandle)

Parameters
pfmHandle
      Handle that came from fmConnect
Return Values
      FM_ST_SUCCESS - successfully disconnected from the FM instance
      FM_ST_UNINITIALIZED - FM interface library has not been initialized
      FM_ST_BADPARAM - if pFmHandle is not a valid handle
      FM_ST_GENERIC_ERROR - an unspecified internal error occurred
```

5.6 Getting Supported Partitions

Use the following FM API to query the list of supported (static) GPU fabric partitions in an NVSwitch based system.

```
fmReturn_t fmGetSupportedFabricPartitions(fmHandle_t pFmHandle,
fmFabricPartitionList_t *pFmFabricPartition)

Parameters
pFmHandle
      Handle returned by fmConnect()
pFmFabricPartition
      Pointer to fmFabricPartitionList_t structure. On success, the list of
supported (static) partition information will be populated in this structure.

Return Values
      FM_ST_SUCCESS – successfully queried the list of supported partitions
      FM_ST_UNINITIALIZED - FM interface library has not been initialized.
      FM_ST_BADPARAM – Invalid input parameters
      FM_ST_GENERIC_ERROR – an unspecified internal error occurred
      FM_ST_NOT_SUPPORTED - requested feature is not supported or enabled
      FM_ST_NOT_CONFIGURED - Fabric Manager is initializing and no data
```

```
        FM_ST_VERSION_MISMATCH - provided versions of params do not match
```

5.7 Activate a GPU Partition

Use the following FM API to activate a supported GPU fabric partition in an NVSwitch based system. This API is supported only in Shared NVSwitch multi-tenancy mode.

```
fmReturn_t fmActivateFabricPartition((fmHandle_t pFmHandle,
fmFabricPartitionId_t partitionId)

 Parameters
pFmHandle
        Handle returned by fmConnect()
partitionId
        The partition id to be activated.

Return Values
        FM_ST_SUCCESS – successfully queried the list of supported partitions
        FM_ST_UNINITIALIZED - FM interface library has not been initialized.
        FM_ST_BADPARAM – Invalid input parameters or unsupported partition id
        FM_ST_GENERIC_ERROR – an unspecified internal error occurred
        FM_ST_NOT_SUPPORTED - requested feature is not supported or enabled
        FM_ST_NOT_CONFIGURED - Fabric Manager is initializing and no data
        FM_ST_IN_USE - specified partition is already active or the GPUs are in
use by other partitions.
```

5.8 Activate a GPU Partition with VFs

Use this function when in vGPU Virtualization Mode, to activate an available GPU fabric partition with vGPU Virtual Functions (VFs). This function must be called before booting VMs using vGPUs including single vGPU VMs. This API is supported only in vGPU multi-tenancy mode.

```
fmReturn_t fmActivateFabricPartitionWithVFs((fmHandle_t pFmHandle,
fmFabricPartitionId_t partitionId, fmPciDevice_t *vfList, unsigned int numVfs)

Parameters:
        pFmHandle
                Handle returned by fmConnect()
        partitionId
                The partition id to be activated.
        *vfList
                List of VFs associated with physical GPUs in the partition. The
ordering of VFs passed to this call is significant, especially for
migration/suspend/resume compatibility, the same ordering should be used each
time the partition is activated.
        numVfs
                Number of VFs

Return Values:
        FM_ST_SUCCESS – successfully queried the list of supported partitions
```

```
      FM_ST_UNINITIALIZED - FM interface library has not been initialized.
      FM_ST_BADPARAM – Invalid input parameters or unsupported partition id
      FM_ST_GENERIC_ERROR – an unspecified internal error occurred
      FM_ST_NOT_SUPPORTED - requested feature is not supported or enabled
      FM_ST_NOT_CONFIGURED - Fabric Manager is initializing and no data
      FM_ST_IN_USE - specified partition is already active or the GPUs are in
use by other partitions.
```

Note:

This API must be called even for a single vGPU partition before launching a vGPU VM.

A multi-vGPU partition activation will fail if the corresponding GPUs have MIG mode enabled on them.

5.9 Deactivate a GPU Partition

Use the following FM API to deactivate a previously activated GPU fabric partition in an NVSwitch based system when Fabric Manager is running in Shared NVSwitch or vGPU multi-tenancy mode.

```
fmReturn_t fmDeactivateFabricPartition((fmHandle_t pFmHandle,
fmFabricPartitionId_t partitionId)

 Parameters
pFmHandle
      Handle returned by fmConnect()
partitionId
      The partition id to be deactivated.

Return Values
      FM_ST_SUCCESS – successfully queried the list of supported partitions
      FM_ST_UNINITIALIZED - FM interface library has not been initialized.
      FM_ST_BADPARAM – Invalid input parameters or unsupported partition id
      FM_ST_GENERIC_ERROR – an unspecified internal error occurred
      FM_ST_NOT_SUPPORTED - requested feature is not supported or enabled
      FM_ST_NOT_CONFIGURED - Fabric Manager is initializing and no data
      FM_ST_UNINITIALIZED - specified partition is not activated
```

5.10 Set Activated Partition List after Fabric Manager Restart

Use the following FM API to send a list of currently activated fabric partitions to Fabric Manager after it has been restarted. This call must be made with number of partitions as zero if there are no active partitions when Fabric Manager is restarted.

```
fmReturn_t fmSetActivatedFabricPartitions(fmHandle_t pFmHandle,
fmActivatedFabricPartitionList_t *pFmActivatedPartitionList)

 Parameters
pFmHandle
       Handle returned by fmConnect()
pFmActivatedPartitionList
       List of currently activated fabric partitions.

Return Values
       FM_ST_SUCCESS – FM state is updated with active partition information
       FM_ST_UNINITIALIZED - FM interface library has not been initialized.
       FM_ST_BADPARAM – Invalid input parameters
       FM_ST_GENERIC_ERROR – an unspecified internal error occurred
       FM_ST_NOT_SUPPORTED - requested feature is not supported or enabled
```

5.11 Get NVLink Failed Devices

Use the following FM API to query all GPUs and NVSwitches with failed NVLinks as part of Fabric Manager initialization.

This API is not supported when Fabric Manager is running in Shared NVSwitch or vGPU multi-tenancy resiliency restart (--restart) modes.

```
fmReturn_t fmGetNvlinkFailedDevices(fmHandle_t pFmHandle,
fmNvlinkFailedDevices_t *pFmNvlinkFailedDevices)

 Parameters
pFmHandle
       Handle returned by fmConnect()
 pFmNvlinkFailedDevices
       List of GPU or NVSwitch devices that have failed NVLinks.

Return Values
       FM_ST_SUCCESS – successfully queried list of devices with failed NVLinks
       FM_ST_UNINITIALIZED - FM interface library has not been initialized.
       FM_ST_BADPARAM – Invalid input parameters
       FM_ST_GENERIC_ERROR – an unspecified internal error occurred
       FM_ST_NOT_SUPPORTED - requested feature is not supported or enabled
```

```
      FM_ST_NOT_CONFIGURED - Fabric Manager is initializing and no data
      FM_ST_VERSION_MISMATCH - provided versions of params do not match
```

5.12 Get Unsupported Partitions

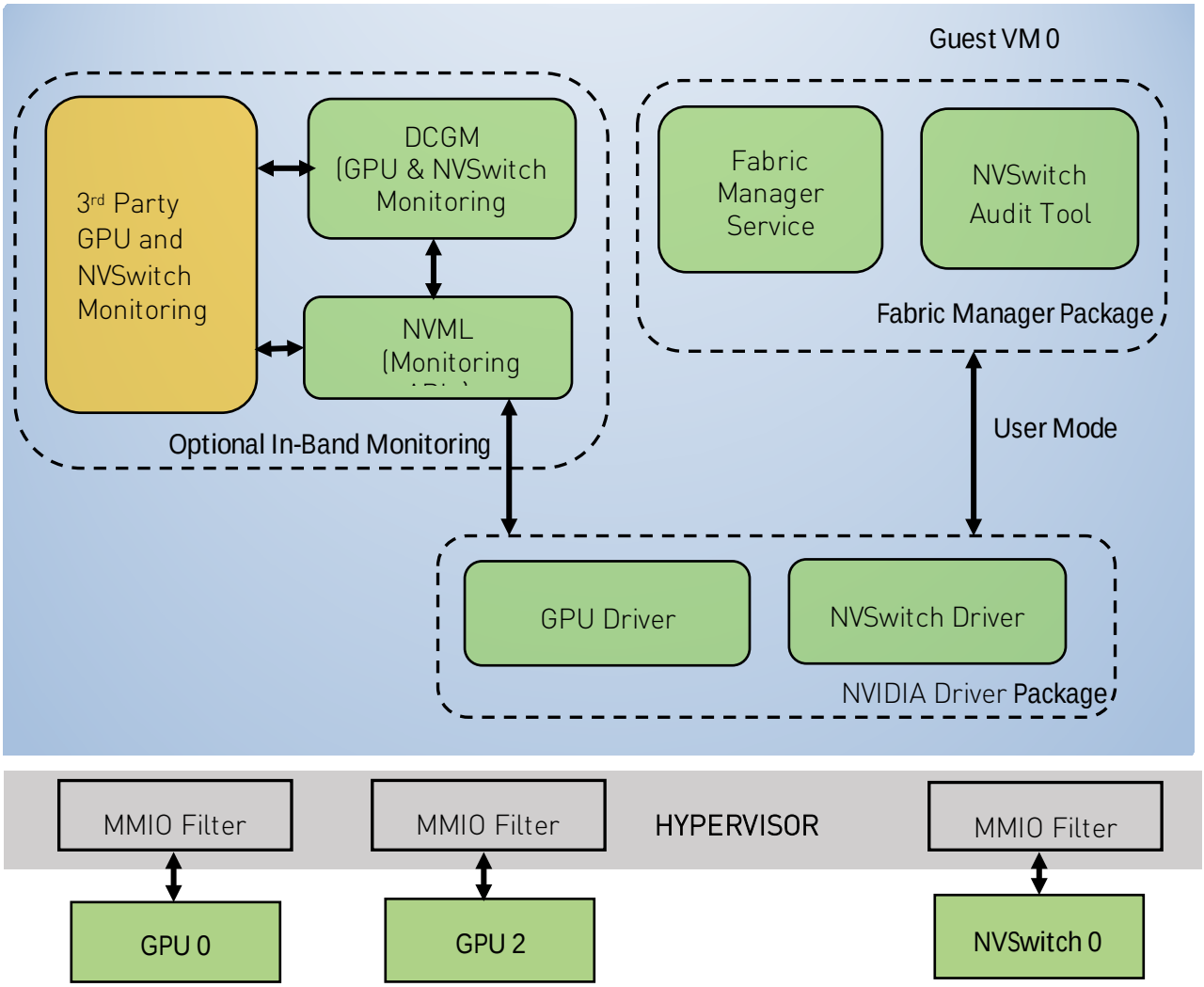
This method is used to query all the unsupported fabric partitions when Fabric Manager is running either in Shared NVSwitch or vGPU multi-tenancy modes.

```
fmReturn_tfmGetUnsupportedFabricPartitions(fmHandle_t pFmHandle,
fmUnsupportedFabricPartitionList_t *pFmUnupportedFabricPartition)
Parameters
pFmHandle
      Handle returned by fmConnect()
pFmUnupportedFabricPartition
      List of unsupported fabric partitions on the system.
Return Values
      FM_ST_SUCCESS – successfully queried list of devices with failed
NVLinks
      FM_ST_UNINITIALIZED - FM interface library has not been
initialized.
      FM_ST_BADPARAM – Invalid input parameters
      FM_ST_GENERIC_ERROR – an unspecified internal error occurred
      FM_ST_NOT_SUPPORTED - requested feature is not supported or
enabled
      FM_ST_NOT_CONFIGURED - Fabric Manager is initializing and no data
      FM_ST_VERSION_MISMATCH - provided versions of params do not match
```

Chapter 6. Full Passthrough Virtualization Model

The first supported virtualization model for NVSwitch-based systems is pass-through device assignment for both GPUs and NVSwitch memory fabrics (switches). VMs with 16, 8, 4, 2 and 1 GPU are supported with predefined subsets of GPUs and switches being used for each VM size. A subset of GPUs and switches is referred to as a system partition. Non-overlapping partitions can be mixed and matched making it possible to support, for example, an 8-GPU VM, a 4-GPU VM and two 2-GPU VMs at the same time on an NVSwitch-based system with two GPU baseboards. VMs with sixteen and eight GPUs have no loss in bandwidth while smaller VMs trade off some bandwidth for isolation by using dedicated switches.

Figure 2. Software Stack within a 2 GPU Virtual Machine: Full Passthrough Model



6.1 Supported VM Configurations

Table 1 Device Assignments with Different VM Configurations

Number of GPUs assigned to a VM	Number of NVSwitch memory fabrics assigned to a VM	Enabled NVLink Interconnects Per GPU	Enabled NVLink Interconnects Per NVSwitch	Constraints
16	12	12 out of 12	32 out of 36	None
8	6	12 out of 12	16 out of 36	One set of eight GPUs from each GPU Baseboard
4	3	6 out of 12	6 out of 36	Two sets of four GPUs from each GPU Baseboard
2	1	2 out of 12	4 out of 36	Four sets of two GPUs from each GPU Baseboard
1	0	0 out of 12	0 out of 36	None

6.2 Virtual Machines with 16 GPUs

All the available GPUs and switches (sixteen GPUs and twelve switches) are assigned to the Guest VM. There are no NVLink interconnects disabled on the switches or on the GPUs. To support 16 GPU partition, the system must be populated with two GPU baseboards.

6.3 Virtual Machines with 8 GPUS

Each VM has eight GPUs and six switches assigned. Each GPU has all 12 NVLink interconnects enabled. If the system is populated with two GPU baseboards, then there will be two available system partitions where 8 GPU VMs can be created, otherwise only one such partition is available. All NVLink connections between GPU baseboards are disabled.

6.4 Virtual Machines with 4 GPUS

Each VM gets four GPUs and three switches. Six NVLink interconnects per GPU are enabled. If the system is populated with two GPU baseboards, then four such partitions

are available on the system. For single baseboard systems, two such partitions are available. All NVLink connections between GPU baseboards are disabled.

6.5 Virtual Machines with 2 GPUs

Each VM gets two GPUs and one NVSwitch. Only two NVLink interconnects per GPU are enabled. If the system is populated with two GPU baseboards, eight such partitions are available on the system. For single baseboard systems, four such partitions are available. All NVLink connections between GPU baseboards are disabled.

6.6 Virtual Machine with 1 GPU

Each VM gets a single GPU and no switches. If the system is populated with two GPU baseboards, 16 such partitions are available on the system. For single baseboard systems, 8 such partitions are available. All NVLink connections between GPU baseboards are disabled.

6.7 Other Requirements

- The hypervisor needs to maintain the partition configuration, including which NVLink connections to block on each GPU and switch for each partition
- The hypervisor needs to implement MMIO filtering for NVSwitch
- The hypervisor needs to finely control IOMMU mappings configured for GPUs and switches
- Guest VMs with more than a single GPU need to run the core NVSwitch software stack (i.e. NVIDIA Driver and Fabric Manager) to configure switches and NVLink connections

6.8 Hypervisor Sequences

The hypervisor needs to flow the following sequence to launch, shutdown and reboot Guest VMs.

- Launching a Guest VM
  - Pick an unused partition of GPUs and switches
  - Reset all the GPUs and switches in the partition
  - Block disabled NVLink connections on each GPU by performing the specified MMIO configuration
  - Block disabled NVLink connections on each switch by configuring the MMIO intercept
  - Avoid configuring any IOMMU mappings among GPUs and switches

- Switches must not be accessible by any other PCIe device that the Guest VM controls, so that they cannot bypass the MMIO restrictions implemented for the CPU
- GPUs do not need to be accessible by any other GPUs or switches
- GPUs need to be accessible by third party devices to support NVIDIA® GPUDirect™ RDMA with them

- Start the Guest VM, avoiding any further GPU resets

- Shutting down a Guest VM
  - Shut down the Guest VM as usual
  - Reset all GPUs and switches that belong to the partition
- Rebooting a Guest VM
  - Need to follow the same steps as launching a Guest VM, except the partition is already chosen

6.9 Monitoring Errors

All the NVSwitch, GPU and NVLink errors are visible to Guest VMs. If the Hypervisor would like to monitor the same, use any of the following methods

- In-band monitoring – Run NVIDIA Data Center GPU Manager (DCGM) on the Guest VM or use the NVIDIA Management Library (NVML) APIs for GPU specific monitoring and NVSwitch public API interface for NVSwitch monitoring.
- Out-of-Band Monitoring – Use the GPU and NVSwitch SMBPBI-based OOB commands.

6.10 Limitations

- NVSwitch errors are visible to the Guest VMs.
- Windows is only supported for single GPU VMs.

Chapter 7. Shared NVSwitch Virtualization Model

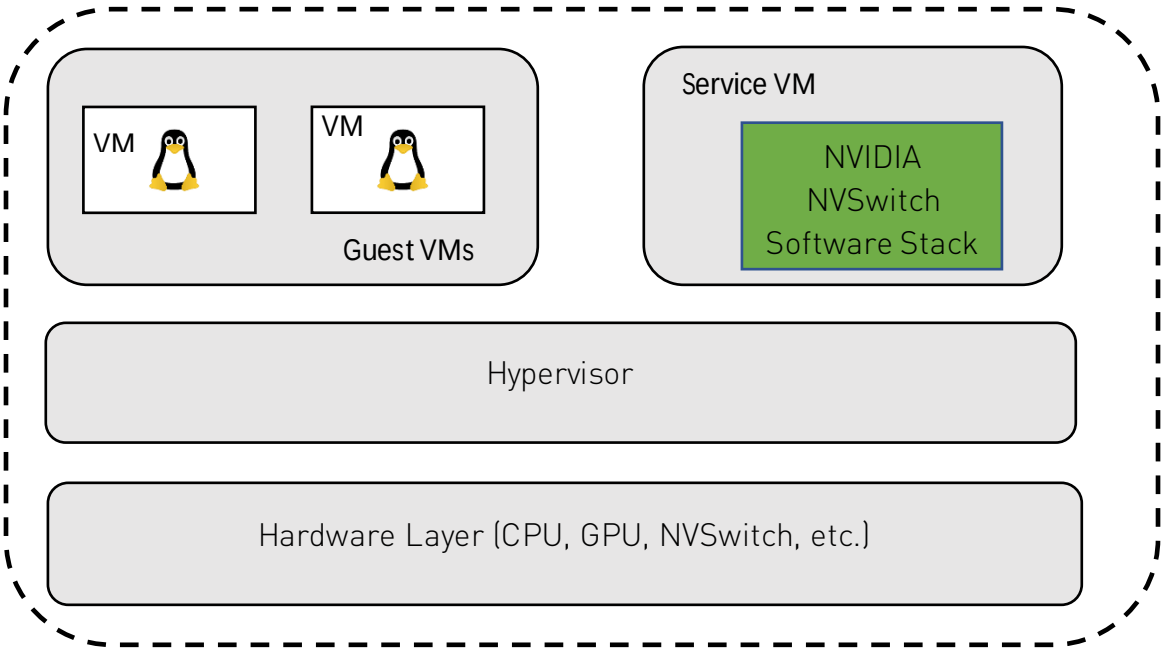
The shared NVSwitch virtualization model extends the GPU Passthrough model further by managing all the switches from a single permanently running Service VM. The GPUs are made accessible to the Service VM for link training and then re-assigned to the Guest VMs.

Sharing switches among all the Guest VMs allows Fabric Manager to enable more NVLink connections for 2 and 4 GPU VMs that observe reduced bandwidth in GPU Passthrough model.

7.1 Software Stack

The software stack required for NVSwitch management runs in a special privileged and trusted VM called the Service VM.

Figure 3. Shared NVSwitch Software Stack



NVSwitch units are always assigned as a PCI pass-through device to the Service VM. GPUs are hot-plugged and hot-unplugged on-demand (as PCI pass-through) to the Service VM. At a high level, Service VM has the following functionalities.

- Provides an interface to query available GPU VM partitions (groupings) and corresponding GPU information.
- Provides an interface to activate GPU VM partitions. This involves
  - Training NVSwitch to NVSwitch NVLink interconnects (if required)
  - Training corresponding GPU NVLink interfaces
  - Programming NVSwitch to deny access to GPUs not assigned to the partition.
- Provides an interface to deactivate GPU VM partitions. This involves the following:
  - Un-train (power down) NVSwitch to NVSwitch NVLink interconnects
  - Un-train (power down) corresponding GPU NVLink interfaces
  - Disable corresponding NVSwitch routing and GPU access.
- Report NVSwitch errors through in-band and out-of-band mechanisms.

7.2 Preparing Service VM

7.2.1 OS Image

NVIDIA internally uses Ubuntu 18.04 LTS as the Service VM OS image. However, there are no known limitations with other major Linux OS distributions. Also, refer to the supported Linux OS compatibility matrix mentioned in chapter 2 for more details.

7.2.2 Resource Requirements

For exact Service VM resource requirements, refer to the corresponding OS distributions minimum resource guidelines. In addition to the specified minimum guidelines, NVIDIA internally uses the following hardware resources for Service VM. Also, this resource requirements may change if Service VM is used for additional functionalities like GPU health check.

Table 2. Service VM Resources

Resource	Quantity/Size
vCPU	2
System Memory	2 GB

7.2.3 NVIDIA Software Packages

The Service VM image must have the following NVIDIA software packages installed.

- NVIDIA Data Center GPU Driver (Version 450.xx and higher)
- NVIDIA Fabric Manager Package (Same version as Driver package)

7.2.4 FM Config File Modifications

To support Shared NVSwitch mode, start the Fabric Manager service in Shared NVSwitch mode by setting the Fabric Manager config item FABRIC_MODE=1.

> **Note:** GPUs and switches must bind to nvidia.ko before Fabric Manager service starts. If the GPUs and switches are not plugged into the Service VM as part of OS boot, start Fabric Manager service manually or the process directly with desired command line options once the GPUs and switches are bound to nvidia.ko.

In Shared NVSwitch mode, Fabric Manager supports a resiliency feature which allows non-stop forwarding of NVLink traffic between GPUs on active Guest VMs after Fabric Manager gracefully or non-gracefully exits in Service VM. To support this feature, Fabric Manager uses /tmp/fabricmanager.state to save certain metadata information. To use a different location/file for storing this metadata information, modify the Fabric Manager config file item STATE_FILE_NAME with the desired path and file name.

Fabric Manager uses TCP I/P loopback (127.0.0.1) based socket interface for communication. To use Unix domain sockets instead, modify the Fabric Manager FM_CMD_UNIX_SOCKET_PATH and UNIX_SOCKET_PATH config file options with the desired Unix domain socket names.

7.2.5 Other NVIDIA Software Packages

In Shared NVSwitch mode, no other process or entity other than Fabric Manager should open and interact with GPUs while activating or deactivating the partition. Also, all the GPU health check applications must be started after activating the partition and must be closed before unbinding the GPUs from nvidia.ko.

7.3 FM Shared Library and APIs

For the list of APIs to manage a shared NVSwitch partition life cycle, refer to Chapter 5 Fabric Manager SDK.

7.3.1 Sample Code

The following code snippet shows how to query supported partitions, activate or deactivate partitions, etc. using the Fabric Manager APIs mentioned in Chapter 5.

```
#include <iostream>
#include <string.h>

#include "nv_fm_agent.h"

int main(int argc, char **argv)
{
    fmReturn_t fmReturn;
    fmHandle_t fmHandle = NULL;
    char hostIpAddress[16] = {0};
    unsigned int operation = 0;
    fmFabricPartitionId_t partitionId = 0;
    fmFabricPartitionList_t partitionList = {0};

    std::cout << "Select Shared Fabric Partition Operation: \n";
    std::cout << "0 - List Supported Partition \n";
    std::cout << "1 - Activate a Partition \n";
    std::cout << "2 - Deactivate a Partition \n";
    std::cin >> operation;
    if ( operation > 2 ) {
        std::cout << "Invalid input.\n" << std::endl;
        return FM_ST_BADPARAM;
    }
    std::cout << std::endl;

    if ( operation > 0 ) {
        std::cout << "Input Shared Fabric Partition ID: \n";
        std::cin >> partitionId;

        if ( partitionId >= FM_MAX_FABRIC_PARTITIONS ) {
            std::cout << "Invalid partition ID." << std::endl;
            return FM_ST_BADPARAM;
        }
    }
    std::cout << std::endl;

    std::cout << "Please input an IP address to connect to. (Localhost =
127.0.0.1) \n";
    std::string buffer;
    std::cin >> buffer;
    if (buffer.length() > sizeof(hostIpAddress) - 1){
        std::cout << "Invalid IP address.\n" << std::endl;
        return FM_ST_BADPARAM;
    } else {
        buffer += '\0';
```

```
        strncpy(hostIpAddress, buffer.c_str(), 15);
    }

    /* Initialize Fabric Manager API interface library */
    fmReturn = fmLibInit();
    if (FM_ST_SUCCESS != fmReturn) {
        std::cout << "Failed to initialize Fabric Manager API interface
library." << std::endl;
        return fmReturn;
    }

    /* Connect to Fabric Manager instance */
    fmConnectParams_t connectParams;
    strncpy(connectParams.addressInfo, hostIpAddress, sizeof(hostIpAddress));
    connectParams.timeoutMs = 1000; // in milliseconds
    connectParams.version = fmConnectParams_version;
    connectParams.addressIsUnixSocket = 0;
    fmReturn = fmConnect(&connectParams, &fmHandle);
    if (fmReturn != FM_ST_SUCCESS){
        std::cout << "Failed to connect to Fabric Manager instance." <<
std::endl;
        return fmReturn;
    }

    if ( operation == 0 ) {
        /* List supported partitions */
        partitionList.version = fmFabricPartitionList_version;
        fmReturn = fmGetSupportedFabricPartitions(fmHandle, &partitionList);
        if (fmReturn != FM_ST_SUCCESS) {
            std::cout << "Failed to get partition list. fmReturn: " << fmReturn
<< std::endl;
        } else {
            /* Only printing number of partitions for brevity */
            std::cout << "Total number of partitions supported: " <<
partitionList.numPartitions  << std::endl;
        }

    } else if ( operation == 1 ) {
        /* Activate a partition */
        fmReturn = fmActivateFabricPartition(fmHandle, partitionId);
        if (fmReturn != FM_ST_SUCCESS) {
            std::cout << "Failed to activate partition. fmReturn: " << fmReturn
<< std::endl;
        }

    } else if ( operation == 2 ) {
        /* Deactivate a partition */
```

```
        fmReturn = fmDeactivateFabricPartition(fmHandle, partitionId);
        if (fmReturn != FM_ST_SUCCESS) {
            std::cout << "Failed to deactivate partition. fmReturn: " <<
fmReturn << std::endl;
        }

    } else {
        std::cout << "Unknown operation." << std::endl;
    }

    /* Clean up */
    fmDisconnect(fmHandle);
    fmLibShutdown();
    return fmReturn;
}

#Make file for the above sample assuming the source is saved into
sampleCode.cpp
# Note: Change the default include paths (/usr/include & /usr/lib) based on FM
API header files location.

IDIR := /usr/include
CXXFLAGS = -I $(IDIR)

LDIR := /usr/lib
LDFLAGS= -L$(LDIR) -lnvfm

sampleCode: sampleCode.o
       $(CXX) -o $@ $< $(CXXFLAGS) $(LDFLAGS)

clean:
       -@rm -f sameplCode.o
       -@rm -f sampleCode
```

7.4 Fabric Manager Resiliency

Refer to Appendix C: Resiliency for detailed information regarding Fabric Manager resiliency in Shared Virtualization mode.

7.5 Cycle Management

7.5.1 GPU Partitions

Refer to Appendix B: GPU Partitions for the default and all supported partitions for the shared NVSwitch virtualization mode.

7.5.2 Building GPUs to Partition Mapping

The Fabric Manager instance running on the Service VM and Hypervisor must use a common numbering scheme (GPU Physical ID) to uniquely identify each GPU. In this release, the Physical ID numbering is the same as in the *HGX-A100 Baseboard Pinout* design collateral.

The Hypervisor should maintain a list of GPU Physical IDs and corresponding PCI BDF mapping information to identify each GPUs within the Hypervisor. This information is required to identify GPUs belonging to a partition and hot-attaching them to Service VM as part of Guest VM activation.

7.5.3 Booting Service VM

As part of Service VM boot, the Hypervisor must do the following.

1. Assign/Plug all the available GPUs and switches as PCI pass-through devices to the Service VM without any MMIO filtering.
2. Start and wait for the Fabric Manager to fully initialize the GPUs and switches. All the Fabric Manager APIs will return FM_ST_NOT_CONFIGURED until the fabric is initialized and ready.
3. Query the list of currently supported VM partitions and build the available Guest VM combinations accordingly.
4. De-assign/Unplug all the GPUs from the Service VM.

7.5.4 Restarting Service VM

Since the NVSwitch kernel software stack gets loaded and initializes the NVSwitches and GPUs as part of the Service VM booting, restarting Service VM will affect already activated GPU partitions. Also, the Hypervisor must follow the same procedure and steps as described in the "Booting Service VM" section.

7.5.5 Shutdown Service

Already activated VM partitions will not be affected as part of Service VM shutdown as NVSwitch configuration is preserved. However, if the Hypervisor issues SBR (Secondary Bus Reset) to NVSwitch devices as part of Service VM shutdown, the activated partitions will be affected. Since Fabric Manager is not running and Driver is unloaded, there will

not be any active error monitoring and corresponding remediation, so it is not recommended to leave the Guest VMs in this state for a longer period.

7.6 Guest VM Life Cycle Management

7.6.1 Guest VM NVIDIA Driver Package

The Guest VM must have the driver package for NVIDIA Data Center GPUs (Version 450.xx or later) installed in order to use GPU NVLink interconnects.

7.6.2 Starting a Guest VM

When starting a Guest VM, the Hypervisor must do the following.

1. Choose one of the supported GPU partitions based on Guest VM GPU demand.
2. Identify the corresponding GPUs using the GPU Physical ID to PCI BDF mapping.
3. Reset (Secondary Bus Reset) all the selected GPUs.
4. Hot plug the selected GPUs to the Service VM (if needed).
5. Make sure the GPUs are bound to nvidia.ko.
6. Request Fabric Manager to activate the requested GPU partition using the fmActivateFabricPartition () API.
7. Unbind the GPUs from nvidia.ko.
8. Hot unplug the GPUs from Service VM (if needed).
9. Launch/start the Guest VM without resetting the GPUs.

7.6.3 Shutting Down a Guest VM

When shutting down a Guest VM, the Hypervisor must do the following.

1. Shut down the Guest VM but avoid any GPU resets (to avoid any NVSwitch side NVLink errors)
2. Request Fabric Manager to deactivate the specific GPU partition using the fmDeactivateFabricPartition () API.
3. Reset the GPUs once the request for Fabric Manager to deactivate the partition is completed.

7.6.4 Rebooting a Guest VM

When rebooting a Guest VM, the Hypervisor must follow the above Guest VM shutdown and Guest VM start sequence if the GPUs get SBR as part of VM reboot.

7.6.5 Verifying GPU Routing

The *nvswitch-audit* command line utility installed as part of the Fabric Manager package can output the number of NVLink connections between each pair of GPUs by reading and decoding internal NVSwitch hardware routing table information. It is recommended to verify the GPU reachability matrix periodically, and on each VM partition activation and deactivation cycle, by running this tool in the Service VM.

The following options are supported by *nvswitch-audit* command line utility.

```
root@host1-servicevm:~# ./nvswitch-audit -h
NVIDIA NVSwitch audit tool
Reads NVSwitch hardware tables and outputs the current number of
NVlink connections between each pair of GPUs

Usage: nvswitch-audit [options]

Options include:
[-h | --help]: Displays help information
[-v | --verbose]: Verbose output including all Request and Response table
entries
[-f | --full-matrix]: Display All possible GPUs including those with no
connecting paths
[-c | --csv]: Output the GPU Reachability Matrix as Comma Separated Values
[-s | --src]: Source GPU for displaying number of unidirectional connections
[-d | --dst]: Destination GPU for displaying number of unidirectional
connections
```

The following output snippet shows the GPU connectivity when an 8 GPU VM partition is activated.

```
root@host1-servicevm:~# ./nvswitch-audit
GPU Reachability Matrix
GPU  1  2  3  4  5  6  7  8
  1  X 12 12 12 12 12 12 12
  2 12  X 12 12 12 12 12 12
  3 12 12  X 12 12 12 12 12
  4 12 12 12  X 12 12 12 12
  5 12 12 12 12  X 12 12 12
  6 12 12 12 12 12  X 12 12
  7 12 12 12 12 12 12  X 12
  8 12 12 12 12 12 12  X 12
```

7.7 Error Handling

Refer to Appendix D: Error Handling for all FM initialization, partition, and hardware specific errors and their handling.

7.7.1 Guest VM GPU Errors

All GPU runtime errors will be logged in the Guest VM syslog as Xid errors when the Guest VM is active. GPU NVLink errors which require re-training are not supported in this environment and must follow the Guest VM shutdown and start sequence to recover.

7.7.2 Handling a Service VM Crash

When a Service VM experiences a kernel crash, all the remaining activated Guest VM should continue as normal. However, the VM partition activation and deactivation life cycle will be affected. A Service VM restart or reboot is required to recover from this state.

7.8 Interoperability With MIG

Shared NVSwitch virtualization model can interoperate with the Multi-Instance GPUs (MIG) feature supported on NVIDIA A100 GPUs. However, the following sequence must be maintained in order to expose a shared NVSwitch partition with MIG enabled GPUs to Guest VMs.

7.8.1 Initializing Service VM

When Fabric Manager initializes on Service VM (without the --restart option for resiliency flow), MIG mode must be disabled for all the available GPUs. If any GPUs have MIG mode enabled, Fabric Manager service initialization will be aborted.

7.8.2 Activating Guest VM

Fabric Manager Shared NVSwitch partition activation and deactivation sequence can handle MIG enabled GPUs. GPUs with MIG enabled before the partition activation (e.g. by VM before doing VM reboot) will not have NVLinks trained as part of the partition activation. The activation/deactivation flow works as is.

Chapter 8. vGPU Virtualization Model

The vGPU virtualization model supports VF pass-through by enabling SR-IOV functionality in all the supported GPUs and assigning a specific VF or set of VFs to the VM.

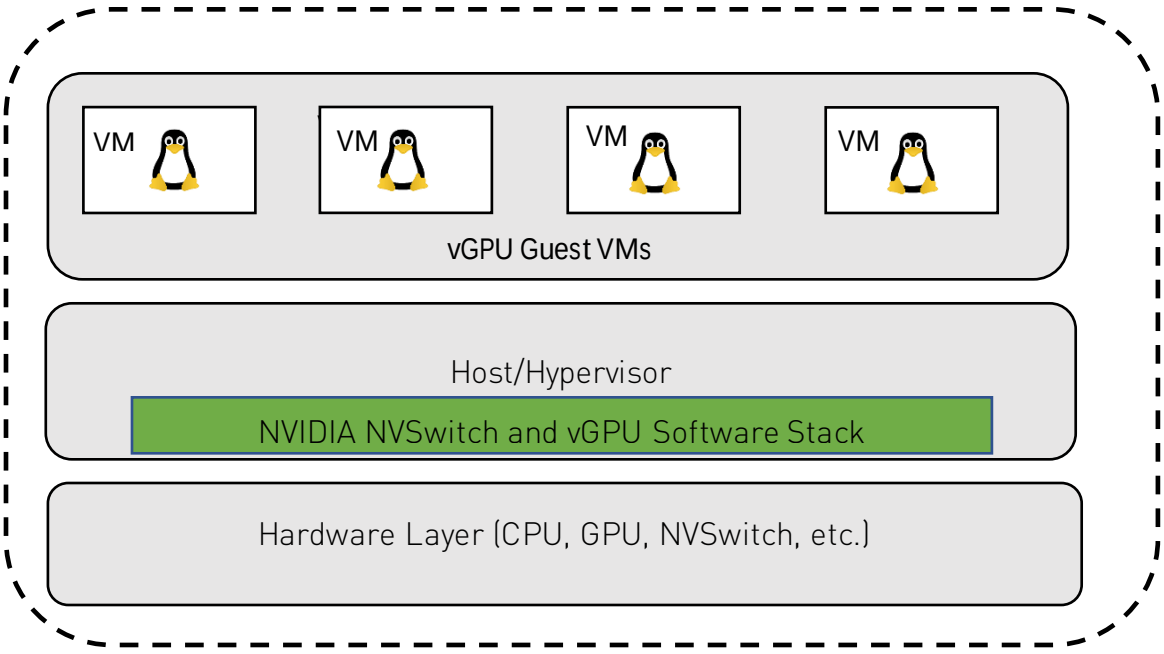
- GPU NVLinks are assigned to only one VF at a time
- NVLink P2P between GPUs belonging to different VMs or partitions is not supported

Refer to the vGPU Software User Guide for all supported vGPU-specific functionality, features, and configurations.

8.1 Software Stack

In the vGPU virtualization model, the NVSwitch Software Stack (FM and Switch Driver) runs in the vGPU host. All the physical GPUs and NVSwitches are owned and managed by the vGPU host as in bare-metal mode. All the GPU and NVSwitch NVLinks are trained and configured as part of Fabric Manager initialization. The switch routing table is initialized to prevent any GPU-GPU communication.

Figure 4. vGPU Software Stack



8.2 Preparing the vGPU Host

8.2.1 OS Image

Refer to the NVIDIA Virtual GPU Software User Guide for the list of supported OSes and Hypervisors, and for instructions on installing and configuring the vGPU host driver software.

8.2.2 NVIDIA Software Packages

In addition to NVIDIA vGPU host driver software, the vGPU host image must have the following NVIDIA software packages installed.

- NVIDIA Fabric Manager Package (must be the same version as the Driver package)
- NVIDIA Fabric Manager SDK Package (must be the same version as the Driver package)

8.2.3 FM Config File Modifications

To support vGPU virtualization, start the Fabric Manager service in vGPU Virtualization mode by setting the Fabric Manager config item FABRIC_MODE=2.

Note: GPUs and NVSwitches must bind to nvidia.ko before Fabric Manager service starts.

In vGPU virtualization mode, Fabric Manager supports a resiliency feature which allows non-stop forwarding of NVLink traffic between GPUs on active guest VMs after Fabric Manager exits (gracefully or non-gracefully) within the vGPU host. To support this feature, Fabric Manager uses /tmp/fabricmanager.state to save certain metadata information. To use a different location/file for storing this metadata information, modify the Fabric Manager config file item STATE_FILE_NAME with the desired path and file name.

By default, Fabric Manager uses TCP I/P loopback (127.0.0.1) based socket interface for communication. To use Unix domain sockets instead, modify the Fabric Manager FM_CMD_UNIX_SOCKET_PATH and UNIX_SOCKET_PATH config file options with the desired Unix domain socket names.

8.3 FM Shared Library and APIs

For the list of APIs to manage vGPU partition life cycle, refer to Chapter 5: Fabric Manager SDK.

8.4 Fabric Manager Resiliency

Refer to Appendix C: Resiliency for detailed information regarding Fabric Manager resiliency in vGPU Virtualization mode.

8.5 vGPU Partitions

Refer to Appendix B: GPU Partitions for the default as well as all supported partitions for the vGPU virtualization model.

8.6 Guest VM Life Cycle Management

The following summarizes the guest VM life cycle.

1. System powers-on and initializes
   a). vGPU host driver loads
   b). SR-IOV is enabled.
   c). FM initializes in vGPU Virtualization Mode
   d). NVlinks are trained.
2. Partition is activated with selected SR-IOV VFs
3. vGPU-enabled VM goes through its life cycle with VFs selected in step 2

   This life cycle can involve boot, reboot, shutdown, suspend, resume, and migrate activities.
4. Partition deactivates

These are explained in more detail in the following sections.

8.6.1 Activating the Partition and Starting the VM

SR-IOV VFs must be enabled on all the physical GPUs before activating partitions and powering on vGPU VMs.

When starting a guest VM, the Hypervisor must do the following.

1. Select an available GPU partition that contains the required number of GPUs for the guest VM, and select the VFs to be used on those GPUs
2. Request Fabric Manager to activate the GPU partition using the `fmActivateFabricPartitionWithVFs` () API, with the set of selected VFs.
3. Start the guest VM with the selected VFs.

> **NOTE**:
> Partition activation is always required before starting a vGPU VM, even for VMs that use only a single vGPU.
>
> The ordering of VFs used during partition activation and VM assignment must remain consistent to ensure correct operation of VM suspend, resume, and migration operations.

Refer to the vGPU User Guide, Section 2.6 Installing and Configuring the NVIDIA GPU Manager for Red Hat Linux KVM for details on SR-IOV VF enablement and assignment of VFs to VMs.

8.6.2 Deactivating the Partition

Partitions should be deactivated only when no VM is executing on the GPUs within the partition. To deactivate a partition:

- Shut down any guest VM currently operating within the partition.
- Request Fabric Manager to deactivate the partition using the fmDeactivateFabricPartition () API.

8.6.3 VM Migration

VM Migration is supported only between partitions with an identical number, type of GPU, and NvLink topology.

Refer to "vGPU User Guide Section 6.3 Migrating a VM Configured with vGPU" for the exact steps.

8.6.4 Verifying GPU Routing

The *nvswitch-audit* command line utility installed as part of the Fabric Manager package can output the number of NVLink connections between each pair of GPUs by reading and decoding internal NVSwitch hardware routing table information. It is recommended to verify the GPU reachability matrix periodically and on each VM partition activation and deactivation cycle, by running this tool in the vGPU host.

The following options are supported by *nvswitch-audit* command line utility.

```
root@host:~# ./nvswitch-audit -h
NVIDIA NVSwitch audit tool
Reads NVSwitch hardware tables and outputs the current number of
NVlink connections between each pair of GPUs

Usage: nvswitch-audit [options]

Options include:
[-h | --help]: Displays help information
[-v | --verbose]: Verbose output including all Request and Response table
entries
[-f | --full-matrix]: Display All possible GPUs including those with no
connecting paths
[-c | --csv]: Output the GPU Reachability Matrix as Comma Separated Values
[-s | --src]: Source GPU for displaying number of unidirectional connections
[-d | --dst]: Destination GPU for displaying number of unidirectional
connections
```

The following output snippet shows the GPU connectivity when an 8 GPU VM partition is activated.

```
root@host1-servicevm:~# ./nvswitch-audit
```

```
GPU Reachability Matrix
GPU  1  2  3  4  5  6  7  8
  1  X 12 12 12 12 12 12 12
  2 12  X 12 12 12 12 12 12
  3 12 12  X 12 12 12 12 12
  4 12 12 12  X 12 12 12 12
  5 12 12 12 12  X 12 12 12
  6 12 12 12 12 12  X 12 12
  7 12 12 12 12 12 12  X 12
  8 12 12 12 12 12 12  X 12
```

8.7 Error Handling

Refer to Appendix D: Error Handling for all FM initialization, partition and hardware specific errors and their handling.

8.7.1 Guest VM GPU Errors

All GPU runtime errors will be logged in the vGPU host syslog as Xid errors when the Guest VM is active. GPU NVLink errors which require re-training are not supported in this environment and must follow the Guest VM shutdown and start sequence to recover.

8.8 GPU Reset

If GPU generates a runtime error or gets NVLink error (Xid), the sysadmin can clear the corresponding error state and recover the GPU using the GPU reset operation. The GPU reset operation must be initiated from the vGPU host once any VM using the GPU is shut down and the corresponding partition is deactivated. Refer to NVIDIA System Management Interface (nvidia-smi) command line utility documentation for more information about GPU reset.

8.9 Interoperability with MIG

MIG-backed vGPUs on NVIDIA A100 cannot use NVlink. Nevertheless, Fabric Manager's vGPU Virtualization mode can interoperate with the Multi-Instance GPUs (MIG) feature, to support use cases where a subset of GPUs are being used in MIG mode.

8.9.1 Enabling MIG before Starting FM Service

- In cases where MIG has been enabled on a GPU prior to starting Fabric Manager, Fabric Manager will remove all GPU partitions that contain GPUs in MIG mode from its list of available partitions.
- These GPU partitions will not be available for deploying any VMs.

- To re-enable partitions after disabling MIG mode on a GPU, reboot the system.

8.9.2 Enabling MIG after Starting FM Service

- MIG functionality may be enabled on any GPU after starting Fabric Manager Service, before any partition contained the GPU is activated.
- Activation of a single GPU partition will return success even if the GPU is in MIG mode.
- Activation of a multi-GPU partition will fail if any GPU within the partition is in MIG mode.

Chapter 9. Supported High Availability Modes

Fabric Manager provides several High Availability Mode (Degraded Mode) configurations which allow system administrators to set appropriate policy when there are hardware failures on NVSwitch based systems such as GPU failures, NVSwitch failures, NVLink connection failures etc. With this feature, system administrators can keep some subset of GPUs available for use while waiting to replace failed GPUs, baseboards etc.

9.1 Definition of Common Terms

- GPU Access NVLink -- This is an NVLink connection between a GPU and a NVSwitch. In DGX A100 & HGX A100, each GPU connects to each NVSwitch over two access NVLinks for a total of 12 per GPU.
- GPU Access NVLink failure -- This failure occurs in the connection between a GPU and an NVSwitch. Failures could be due to a GPU/NVSwitch pin failure, mechanical failure in the GPU baseboard, or similar.
- Trunk NVLink -- These are the links that connect two GPU baseboards. Trunk NVLinks only occur between NVSwitches and travel over the NVLink bridge PCBs and connectors.
- Trunk NVLink failure -- A trunk NVLink failure is a failure in the NVLinks that traverse between the two GPU baseboard trays. This failure could be due to a bad backplane connector pin or similar.
- NVSwitch failure -- A NVSwitch failure is categorized as an internal failure of the NVSwitch itself. This failure could be due to the NVSwitch not showing on the PCIe bus, DBE error or similar.
- GPU failure -- A GPU failure is where the GPU itself has failed in some manner. This could be NVLink connectivity, PCIe failure, or similar.

) Note: All these high availability modes and their corresponding dynamic reconfiguration of the NVSwitch based system are applied in response to errors detected during Fabric Manager initialization. Any run-time errors happening after the system is initialized or when a GPU job is running will not trigger any of these high availability mode policies.

9.2 GPU Access NVLink Failure

9.2.1 Fabric Manager Config Item

GPU access NVLink failure mode is controlled through this Fabric Manager config file item:

```
ACCESS_LINK_FAILURE_MODE=<value>
```

9.2.2 Bare Metal Behavior

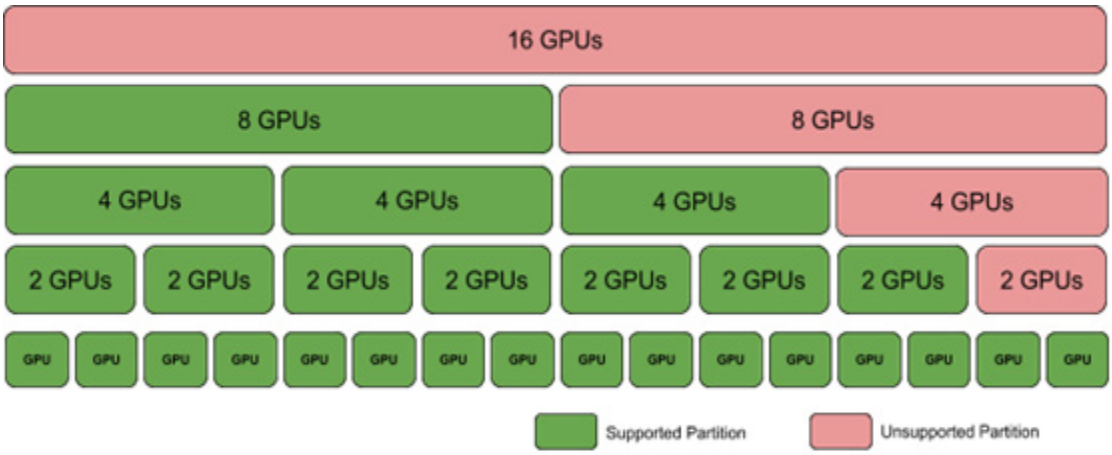
- ACCESS_LINK_FAILURE_MODE=0

In this mode, Fabric Manager will remove the GPUs with access NVLink failure from NVSwitch routing and configure the rest of the GPUs to form a single memory fabric. This means those access NVLink failed GPUs will lose their NVLink peer-to-peer (P2P) capability with other GPUs. The failed GPUs are still visible to the NVIDIA software stack (like CUDA, NVML, NVIDIA-SMI etc.) and can be used for Non-NVLink workload.

- ACCESS_LINK_FAILURE_MODE=1

In this mode, Fabric Manager will disable the NVSwitch (and its pair of Trunk NVLinks in case of two GPU baseboards) where the GPU Access NVLink is connected, reducing the NVLink P2P bandwidth to

Figure 4. Shared NVSwitch and vGPU partitions when a GPU Access NVLink fails



- ACCESS_LINK_FAILURE_MODE=1

In the Shared NVSwitch mode, all GPU partitions will be available but will reduce the available bandwidth to throughout the fabric. If multiple access NVLinks fail on a single GPU, then that GPU will be removed and the available GPU partitions will be adjusted as mentioned above. In any case, the failed GPUs will be available for single GPU partitions.

) **Note:** Currently ACCESS_LINK_FAILURE_MODE=1 configuration is not supported in vGPU Multitenancy Mode.

9.3 Trunk NVLink Failure

9.3.1 Fabric Manager Config Item

Trunk NVLink failure mode is controlled through this Fabric Manager config file item:

TRUNK_LINK_FAILURE_MODE=<value>

) Note: This option is applicable only to systems with two GPU baseboards.

9.3.2 Bare Metal Behavior

- TRUNK_LINK_FAILURE_MODE=0

In this mode, Fabric Manager will abort and leave the system uninitialized when there is a trunk NVLink failure and all CUDA application launches will fail with *cudaErrorSystemNotReady* status. However, when the continue with error config option (FM_STAY_RESIDENT_ON_FAILURES =1) is enabled, Fabric Manager service will continue to run and CUDA application launches will fail with *cudaErrorSystemNotReady* status.

- TRUNK_LINK_FAILURE_MODE=1

In this mode, if an NVSwitch has a single or multiple trunk NVLink failures, then that specific NVSwitch will be disabled along with its peer NVSwitch. This will reduce the available bandwidth to throughout the fabric. If multiple NVSwitches have trunk NVLink failures, then Fabric Manager will fall back to TRUNK_LINK_FAILURE_MODE=0 behavior as mentioned above.

9.3.3 Shared NVSwitch and vGPU Virtualization Behavior

- TRUNK_LINK_FAILURE_MODE=0

In these modes, Fabric Manager will remove GPU partitions using trunk NVLinks from the currently supported GPU partition list. This means 16 GPU partitions and 8 GPU partitions across baseboards will be removed. All the remaining partitions will run with full NVLink bandwidth. This option will support any number of trunk NVLink failures on a single connected pair of NVSwitches.

- TRUNK_LINK_FAILURE_MODE=1

In the Shared NVSwitch mode, all the GPU partitions will be available but will reduce the available bandwidth to throughout the fabric. This option will be supported if multiple trunk NVLink failures are present on the same NVSwitch pair. If multiple trunk NVLink failures affect different NVSwitch pairs then Fabric Manager will fall back to TRUNK_LINK_FAILURE_MODE=0 behavior as mentioned above.

) **Note:** Currently TRUNK_LINK_FAILURE_MODE=1 configuration is not supported in vGPU Multitenancy Mode.

9.4 NVSwitch Failure

9.4.1 Fabric Manager Config Item

NVSwitch failure mode is controlled through this Fabric Manager config file item:

```
NVSWITCH_FAILURE_MODE=<value>
```

9.4.2 Bare Metal Behavior

- NVSWITCH_FAILURE_MODE=0

In this mode, Fabric Manager will abort and leave the system uninitialized when there is an NVSwitch failure and all CUDA application launches will fail with *cudaErrorSystemNotReady* status. However, when the continue with error config option (FM_STAY_RESIDENT_ON_FAILURES =1) is enabled, Fabric Manager service will continue to run and CUDA application launches will fail with *cudaErrorSystemNotReady* status.

- NVSWITCH_FAILURE_MODE =1

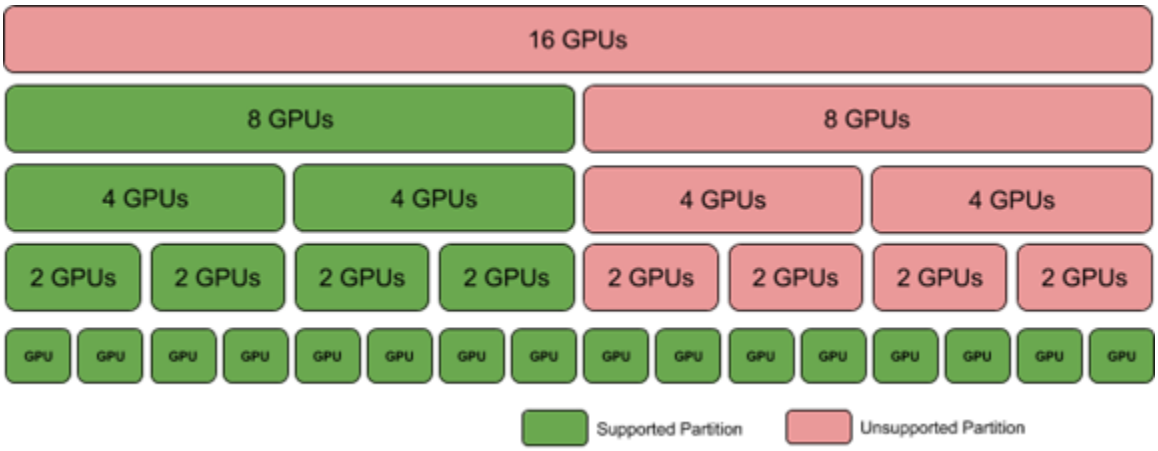
In this mode, when there is an NVSwitch failure, that specific NVSwitch will be disabled along with its peer NVSwitch. This will reduce the available bandwidth to throughout the fabric. If multiple NVSwitch failures happen, then Fabric Manager will fall back to the NVSWITCH_FAILURE_MODE=0 behavior as mentioned above.

9.4.3 Shared NVSwitch and vGPU Virtualization Behavior

- NVSWITCH_FAILURE_MODE=0

In these modes, Fabric Manager will remove multi-GPU partitions from the baseboard with the failing NVSwitch and 8 GPU partitions across baseboards. This means, in a single baseboard system only single GPU partitions will be supported. The following picture shows supported partitions when an NVSwitch has failed.

Figure 5. Shared NVSwitch and vGPU partitions when an NVSwitch has failed



- NVSWITCH_FAILURE_MODE=1

In the Shared NVSwitch mode, all the GPU partitions will be available but will reduce the available bandwidth to throughout the fabric. If multiple NVSwitch failures happen, Fabric Manager will fall back to NVSWITCH_FAILURE_MODE =0 behavior as mentioned above.

) **Note:** Currently NVSWITCH_FAILURE_MODE=1 configuration is not supported in vGPU Multitenancy Mode.

9.5 GPU Failure

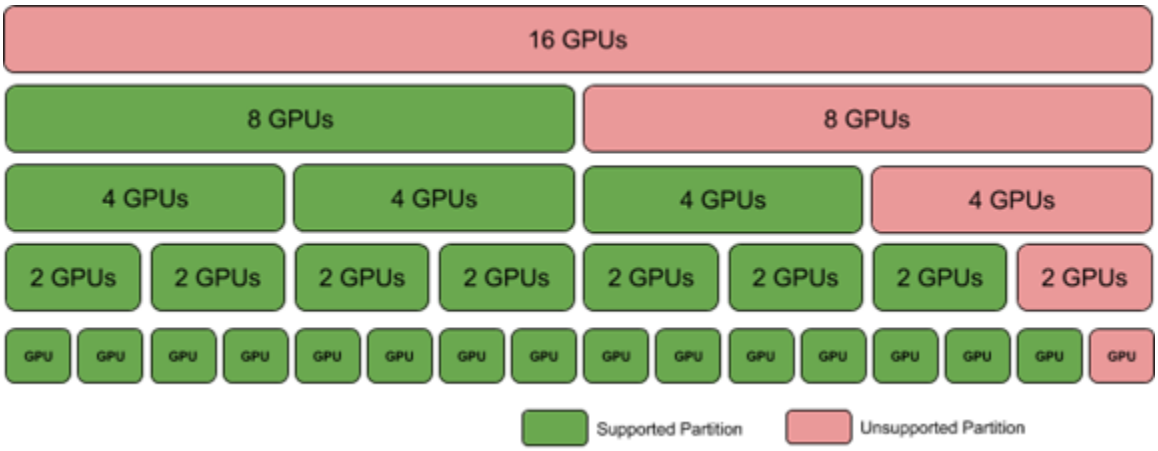
9.5.1 Bare Metal Behavior

Fabric Manager will ignore GPUs which have failed to initialize or are missing (due to not showing on the PCI bus) etc. Fabric Manager will set up routing and enable NVLink P2P among available GPUs.

9.5.2 Shared NVSwitch and vGPU Virtualization Behavior

Fabric Manager will continue initialization and will adjust the currently supported partition list by excluding the failed GPU partitions. The following picture shows supported partitions when a GPU is missing or failed to initialize.

Figure 6. Shared NVSwitch and vGPU partitions when a GPU is missing/failed



9.6 Manual Degradation

The following mechanism is available to prevent a consistently failing GPU, NVSwitch or baseboard is being enumerated by NVSwitch system software stack. These mechanisms require system administrator's intervention to configure appropriate action depending on the failing component.

9.6.1 GPU Exclusion

Depending on the errors, certain GPUs may be candidates for exclusion from the system, so that Fabric Manager can successfully initialize and configure the rest of the GPU subsets. Based on the failure analysis data from previous generation GPUs, the following 5 error conditions are recommended candidates for excluding a GPU.

- GPU double bit ECC errors
- GPU falling off the PCIe bus
- GPU failure to enumerate on PCIe bus
- GPU side NVLink training error
- GPU side unexpected XID: This category can also be application induced

For full passthrough virtualization, the expectation is that the administrator first identifies GPUs that should be blacklisted. The responsibility is on the hypervisor to ensure that VMs are not created on the GPUs that are identified as candidates for exclusion.

9.6.1.1 GPU Exclusion Flow

The GPU exclusion flow can be broken down into 3 different phases.

- Running application error handling
- Diagnosis of GPU failures
- Remediation of error

The precise steps for each of these phases can vary based on whether the system is running in bare metal or in virtualized mode. The following sections describe the flow for bare metal and virtualized platforms in detail.

9.6.1.2 Running Application Error Handling

Errors hit by the GPU during active execution (e.g. GPU ECC errors, GPU falling off the bus) are reported through any of the following,

- /var/log/syslog as an XID message
- NVIDIA Data Center GPU Manager (DCGM)
- NVIDIA Management Library (NVML)
- GPU SMBPBI based OOB commands
- Fabric Manager log file.

Error Condition	Error signature on running application
GPU Double Bit Error	XID 48 output by GPU driver
GPU falling off PCIe bus	XID 79 output by GPU driver
GPU failing to enumerate on bus	GPU does not appear to applications (CUDA applications or nvidia-smi query)
GPU side NVLink training error	Error output to /var/log/syslog by Fabric Manager
GPU side errors	Other XIDs output by GPU driver. This can also be application induced.
GPU Double Bit Error	XID 48 output by GPU driver

9.6.1.3 Diagnosis of GPU Failures

System administrators can create their own GPU monitoring/health check scripts to look for the above-mentioned error traces. This will require looking at one or more of the above-mentioned sources (syslog, NVML APIs, etc.) to collect the desired data.

NVIDIA DCGM includes a built-in *blacklist recommendation script* which can be invoked by a system administrator to collect the above GPU error information. This script queries information from the passive monitoring performed by DCGM to determine if any condition that may require a GPU to be blacklisted has occurred since the last time the DCGM daemon was started. The DCGM recommendation script also invokes a validation

test as part of its execution. This validation test can determine if unexpected XIDs are being generated by the execution of a known good application. If the user desires, they can prevent the validation test from being run, and choose to only monitor the passive information.

The DCGM blacklist recommendation script code is provided as a reference for system administrators to extend as appropriate or build their own monitoring/health check scripts.

Note: Refer to the for more information about the blacklist recommendation script such as its location and supported options.

9.6.1.4 In-Band GPU Blacklisting Mechanism

The GPU kernel driver on NVSwitch based systems can be configured to ignore a set of GPUs, even if they were enumerated on the PCIe bus. Candidate GPUs to be blacklisted are identified by the GPU's unique identifier (GPU UUID) via a kernel module parameter. The GPU kernel module driver will identify if any of the GPU blacklist candidates were present in the system, and if so, will blacklist the GPU from usage by applications. If a GPU UUID is in the blacklist candidate list, but that UUID is not detected at runtime, either due to the UUID belonging to a GPU not on the system, or due to PCIe enumeration of the GPU board failing, the GPU is not considered to have been blacklisted.

The list of blacklist candidate GPUs can be persisted across reboots by specifying the module parameters using a .conf file in the filesystem. The blacklisting mechanism is specific to a given GPU, rather than physical location on the baseboard. Thus, if a GPU is on the blacklist candidate list, and is later replaced by a new GPU, the new GPU will become visible to the system without needing any updates to the blacklist candidates. Conversely if a GPU has been blacklisted on a system, placing it in different PCIe slots will still prevent the GPU from being made visible to applications, unless the blacklist candidate list is updated.

The update of the GPU blacklist candidates is a step that requires manual intervention by the system administrator.

9.6.1.5 Kernel Module Parameters

The set of candidate GPU UUIDs that are to be blacklisted are specified via a kernel module parameter consisting of a set of comma-separated GPU UUIDs. The kernel parameter can be specified at the time of the kernel module load of nvidia.ko as follows,

```
insmod nvidia.ko NVreg_gpuBlacklist=uuid1,uuid2…
```

To make the GPU UUID persistent, the set of blacklist candidate GPU UUIDs can also be specified via an nvidia.conf file in /etc/modprobe.d as follows,

```
options nvidia NVreg_GpuBlacklist=uuid1, uuid2…
```

The addition of GPUs into the blacklist candidate list is a manual step performed by a system administrator.

9.6.1.6 Adding/removing a GPU from the Blacklist Candidate List

To add or remove a GPU into the blacklist candidate list, the system administrator must perform the following manual steps

1. Create a new .conf file for the nvidia kernel module parameters (if none exists)
2. For additions, add the UUID of the to be blacklisted GPU into the .conf file. For removal, remove the UUID of the GPU from the list
3. Restart the system to load the kernel driver with updated module parameters.

9.6.1.7 Listing Blacklisted GPUs

A blacklisted GPU is not visible in CUDA applications or in basic queries using "nvidia-smi -q", or through NVML. However, there are 2 new mechanisms to identify when a GPU has been blacklisted (i.e., the GPU's UUID was present in the blacklist candidate list and the GPU was detected in the system).

9.6.1.8 nvidia-smi

The new command `nvidia-smi -B` or `nvidia-smi --list-blacklist-gpus` can be used to get a list of the blacklisted GPUs

9.6.1.9 Procfs

The procfs entry `/proc/driver/nvidia/gpus/<PCI_ID>/information` can specify whether the GPU in question has been blacklisted.

9.6.1.10 Out-of-Band Query

Refer to the NVIDIA GPU SMBus Post-Box Interface (SMBPBI) Documentation.

9.6.1.11 Running GPU Exclusion Scripts

The following section details recommended flow that a system administrator should follow to run their GPU monitoring health checks or the DCGM exclusion recommendation script on various system configurations.

9.6.1.12 Bare metal and vGPU Configurations

Both in Bare metal and vGPU virtualization configurations, the system administrator is running in the same OS instance as the application programs. The general flow that a system administrator should follow is:

1. Periodically run the health check script or DCGM blacklist recommendation script for all the GPUs and NVSwitches on the system.
2. Optionally, monitor the system logs to trigger a run of the health check script or DCGM blacklist recommendation script.
3. Based on the output of the health check or blacklist recommendation script, add the GPU UUID to the blacklist candidate list.

   Also, if using the DCGM blacklist recommendation script, update the periodic run of the blacklist recommendation script with the newly expected GPU count.
4. Reboot the system to load the kernel driver with updated module parameters

9.6.1.13 Full Pass through Virtualized Configurations

The primary difference on virtualized configurations is that the GPU kernel driver is left to the Guest VMs. As a result, the execution of GPU diagnosis and remediation phases must be performed by the hypervisor in conjunction with the VM provisioning mechanism.

The general flow a hypervisor should follow is:

1. The Guest VM finishes and returns controls of a set of GPUs and switches to the hypervisor
2. The hypervisor invokes a special test VM, which is trusted by the hypervisor.
3. Within the test VM, there should be a full instance of the NVIDIA NVSwitch core software stack (i.e., GPU drivers and FM).
4. On this test VM, run the health check script or DCGM blacklist recommendation script.
5. Based on the output of the health check or blacklist recommendation script, add the GPU UUID to a hypervisor readable database.
6. Hypervisor shuts down the test VM.
   a). The hypervisor reads the database to identify the candidates for blacklisting and updates its resource allocation mechanisms to prevent that GPU from being assigned to future VM requests.
   b). When the GPU board in question has been replaced, the hypervisor updates the database to make the GPU available for access.

9.6.1.14 Shared NVSwitch Virtualization Configurations

In shared NVSwitch virtualization configuration, the system administrator can run their GPU health check script or DCGM exclusion recommendation script in a dedicated test VM or on the Service VM immediately after the GPU partition is activated.

Running GPU health on special test VM

- The Guest VM finishes and returns control of the GPUs in the partition to the hypervisor
- Once the shared NVSwitch Guest VM shutdown procedure is completed, activate the same GPU partition again.
- Then the hypervisor schedules a special test VM which is trusted on those GPUs.
- On this test VM, run the health check script or DCGM exclusion recommendation script.
- Based on the output of the health check or exclusion recommendation script add the GPU UUID into a hypervisor readable database.
- The hypervisor can consider adding all the GPU UUID s of a partition to the database if that partition activation/deactivation cycle is consistently failing.
- Once the health check is done, shutdown the test VM.
- The hypervisor should read the database to identify the candidates for blacklisting and should remove corresponding GPU partitions from its currently supported partitions.
- The hypervisor resource allocation mechanisms should ensure that the affected GPU partitions will not be activated.
- When the Service VM is rebooted, the hypervisor can choose not to bind those blacklisted GPUs to the Service VM. This way Fabric Manager will adjust its currently supported GPU partitions accordingly.
- When the GPU board in question has been replaced, the hypervisor should update the database to make the GPU available and restart the Service VM with all the GPUs to re-enable previously disabled GPU partitions.

Running GPU health on Service VM

- The fmActivateFabricPartition() call returned successfully in a Shared NVSwitch partition activation flow.
- Before the hypervisor detaches/unbinds the GPUs in the partition, run the required health check script or DCGM blacklist recommendation script on those GPUs in the Service VM.
- Based on the output of the health check or blacklist recommendation script add the GPU UUID into a hypervisor readable database.

- The hypervisor must execute the partition deactivation flow using fmDeactivateFabricPartition() when health check fails and corresponding guest VM launch is deferred.
- The hypervisor can consider adding all the GPU UUID s of a partition to the database if that partition activation/deactivation cycle is consistently failing.
- The hypervisor should then read the database to identify the candidates for blacklisting and should remove corresponding GPU partitions from its currently supported partitions.
- The hypervisor resource allocation mechanisms should ensure that the affected GPU partitions will not be activated.
- When the Service VM is rebooted, the hypervisor can choose not to bind those blacklisted GPUs to the Service VM. This way Fabric Manager will adjust its currently supported GPU partitions accordingly.
- When the GPU board in question has been replaced, the hypervisor should update the database to make the GPU available and restart Service VM with all the GPUs to re-enable previously disabled GPU partitions.

9.6.2 NVSwitch Exclusion

In DGX A100 and NVIDIA HGX A100 systems, if an NVSwitch is consistently failing, the system administrator can explicitly exclude the specified NVSwitch using the following mechanisms.

9.6.2.1 In-Band NVSwitch Exclusion

The NVSwitch kernel driver on NVSwitch-based systems can be configured to ignore an NVSwitch even if they were enumerated on the PCIe bus, similar to the GPU exclusion feature mentioned above. The NVSwitch kernel module driver will identify if any of the NVSwitch exclusion candidates were present in the system, and if so, will exclude the NVSwitch from usage by applications. If an NVSwitch UUID is in the exclusion candidate list but that UUID is not detected at runtime, either because the UUID belongs to a NVSwitch not on the system or because PCIe enumeration of the NVSwitch fails, the NVSwitch is not considered to have been excluded.

Also, in HGX A100 systems with two GPU baseboards, if an NVSwitch is explicitly excluded then Fabric Manager will manually exclude its peer NVSwitch across the Trunk NVLinks. This behavior can be configured using the NVSWITCH_FAILURE_MODE high availability configuration file item.

9.6.2.2 Kernel module parameters

The candidate NVSwitch UUID can be specified as a kernel module parameter as follows.

```
insmod nvidia.ko NvSwitchBlacklist=<NVSwitch_uuid>
```

To make the NVSwitch UUID persistent, specify the same via an nvidia.conf file in /etc/modprobe.d as follows,

```
options nvidia NvSwitchBlacklist=<NVSwitch_uuid>
```

The system administrator can get NVSwitch UUID from the Fabric Manager log file. The addition of NVSwitch UUID into the excluded candidate list is a manual step performed by a system administrator.

9.6.2.3 Out-of-Band NVSwitch Exclusion

Refer to SMBus Post Box Interface (SMBPBI) for NVSwitch Documentation.

Appendix A. NVLink Topology

A.1 HGX A100 GPU Baseboard

A list of link IDs used by each GPU to connect to each NVSwitch is provided in the following table. Every NVSwitch uses links 0 to 7 and 16 to 23 for the inter-GPU baseboard connection and they are not listed. Other NVLink connections (four per NVSwitch) are unused.

The GPU numbering in the following table is the same numbering used in the *HGX A100 Baseboard Pinout* design collateral.

Table 3. NVLink Topology of the HGX A100 GPU Baseboard

GPU	GPU link	NVSwitch	NVSwitch link
1	0, 1	4	8, 9
1	2, 3	1	24, 25
1	4, 5	3	30, 31
1	6, 7	6	12, 13
1	8, 9	2	12, 13
1	10, 11	5	30, 31
2	0, 1	4	30, 31
2	2, 3	1	26, 27
2	4, 5	3	12, 13
2	6, 7	6	24, 25
2	8, 9	2	34, 35
2	10, 11	5	14, 15
3	0, 1	4	28, 29
3	2, 3	1	34, 35
3	4, 5	3	34, 35
3	6, 7	6	26, 27
3	8, 9	2	8, 9
3	10, 11	5	12, 13

GPU	GPU link	NVSwitch	NVSwitch link
4	0, 1	4	34, 35
4	2, 3	1	32, 33
4	4, 5	3	14, 15
4	6, 7	6	14, 15
4	8, 9	2	14, 15
4	10, 11	5	34, 35
5	0, 1	4	26, 27
5	2, 3	1	10, 11
5	4, 5	3	28, 29
5	6, 7	6	28, 29
5	8, 9	2	10, 11
5	10, 11	5	26, 27
6	0, 1	4	10, 11
6	2, 3	1	28, 29
6	4, 5	3	8, 9
6	6, 7	6	10, 11
6	8, 9	2	24, 25
6	10, 11	5	28, 29
7	0, 1	4	24, 25
7	2, 3	1	30, 31
7	4, 5	3	26, 27
7	6, 7	6	30, 31
7	8, 9	2	26, 27
7	10, 11	5	10, 11
8	0, 1	4	32, 33
8	2, 3	1	8, 9
8	4, 5	3	10, 11
8	6, 7	6	34, 35
8	8, 9	2	32, 233
8	10, 11	5	24, 25

Appendix B. GPU Partitions

B.1 DGX A100 and HGX A100

B.1.1 Default GPU Partitions

The default system partitions for Shared NVSwitch and vGPU model are shown in Table 1 for DGX A100 and NVIDIA HGX A100 systems with two GPU baseboards. Depending on the high availability mode configurations, when a GPU is not available (due to failures, backlisting etc.), the corresponding partitions will be removed from the currently supported partition list. However, the Partition ID and GPU Physical IDs will remain the same for the rest of the supported partitions.

Note: The GPU Physical IDs are based on how the GPU baseboard NVSwitch GPIOs are strapped. If only one baseboard is present and the GPIOs are strapped as for the bottom tray, then the GPU Physical IDs range from 1 to 8. If the baseboard is strapped as for the top tray, then the GPU Physical IDs range from 9 to 16.

Table 1. Default Shared NVSwitch and vGPU partitions for DGX A100 and NVIDIA HGX A100

Partition ID	Number of GPUs	GPU Physical ID	Number of NVLink Interconnects per GPU
0	16	1 to 16	12
1	8	1 to 8	12
2	8	9 to 16	12
3	8	1 to 4 & 9 to 12	12
4	8	5 to 8 & 13 to 16	12
5	8	1 to 4 & 13 to 16	12
6	8	5 to 8 & 9 to 12	12
7	4	1, 2, 3, 4	12
8	4	5, 6, 7, 8	12

Partition ID	Number of GPUs	GPU Physical ID	Number of NVLink Interconnects per GPU
9	4	9, 10, 11, 12	12
10	4	13, 14, 15, 16	12
11	2	1, 2	12
12	2	3, 4	12
13	2	5, 6	12
14	2	7, 8	12
15	2	9, 10	12
16	2	11, 12	12
17	2	13, 14	12
18	2	15, 16	12
19	1	1	0
20 to 34	1	Physical ID 2 for Partition ID 20, Physical ID 3 for Partition ID 21, etc.	0

B.1.2 Supported GPU Partitions

In DGX A100 and HGX A100 systems, the earlier generation even-odd pair NVSwitch NVLink reset requirement is no longer applicable. So, if the default GPU partition mentioned above is not optimal based on the system's PCIe topology, the partition mapping can be changed. However, NVIDIA has the following restrictions for partition definitions.

- All the 2 GPU NVLink partitions must be within the same GPU baseboard
- All the 4 GPU NVLink partitions must be within the same GPU baseboard
- For 8 GPU NVLink partitions, which span across two GPU baseboards, 4 GPUs must be from each baseboard.

) Note: NVIDIA will evaluate any custom partition definition requests and any variations of the above policy on a case by case basis and will provide necessary information to configure/override the default GPU partitions.

Appendix C. Resiliency

The Fabric Manager resiliency feature in Shared NVSwitch and vGPU Model allows system administrators to resume the normal operation after Fabric Manager gracefully or non-gracefully exits in the Service VM. With this feature, already activated Guest VMs will continue to forward NVLink traffic when FM is not running. Once successfully restarted, Fabric Manager will support the normal Guest VM activation /deactivation workflow.

All the NVSwitch and GPU NVLink errors detected (if any) while Fabric Manager is not running will be cached into the NVSwitch Driver and will be reported once Fabric Manager is successfully restarted. Also, changing the Fabric Manager version in-between is not supported.

C.1 High Level Flow

- After a Fabric Manager crash or graceful exit, Hypervisor should start Fabric Manager with a special command line option (--restart) to resume the operation.
- After restarting Fabric Manager with desired options, the Hypervisor should provide a list of currently activated Guest VM partitions by using fmSetActivatedFabricPartitions () API within 60 seconds. This is because Fabric Manager has no knowledge of Guest VM changes when it is not running. Hypervisor should call the fmSetActivatedFabricPartitions () API with number of partitions as zero if there are no activated Guest VM partitions running when FM is restarted.
- To start Fabric Manager normally or to reinitialize the software and hardware states, Hypervisor should follow the normal Service VM starting sequence. (i.e., start without the --restart command line option)

C.2 Detailed Resiliency Flow

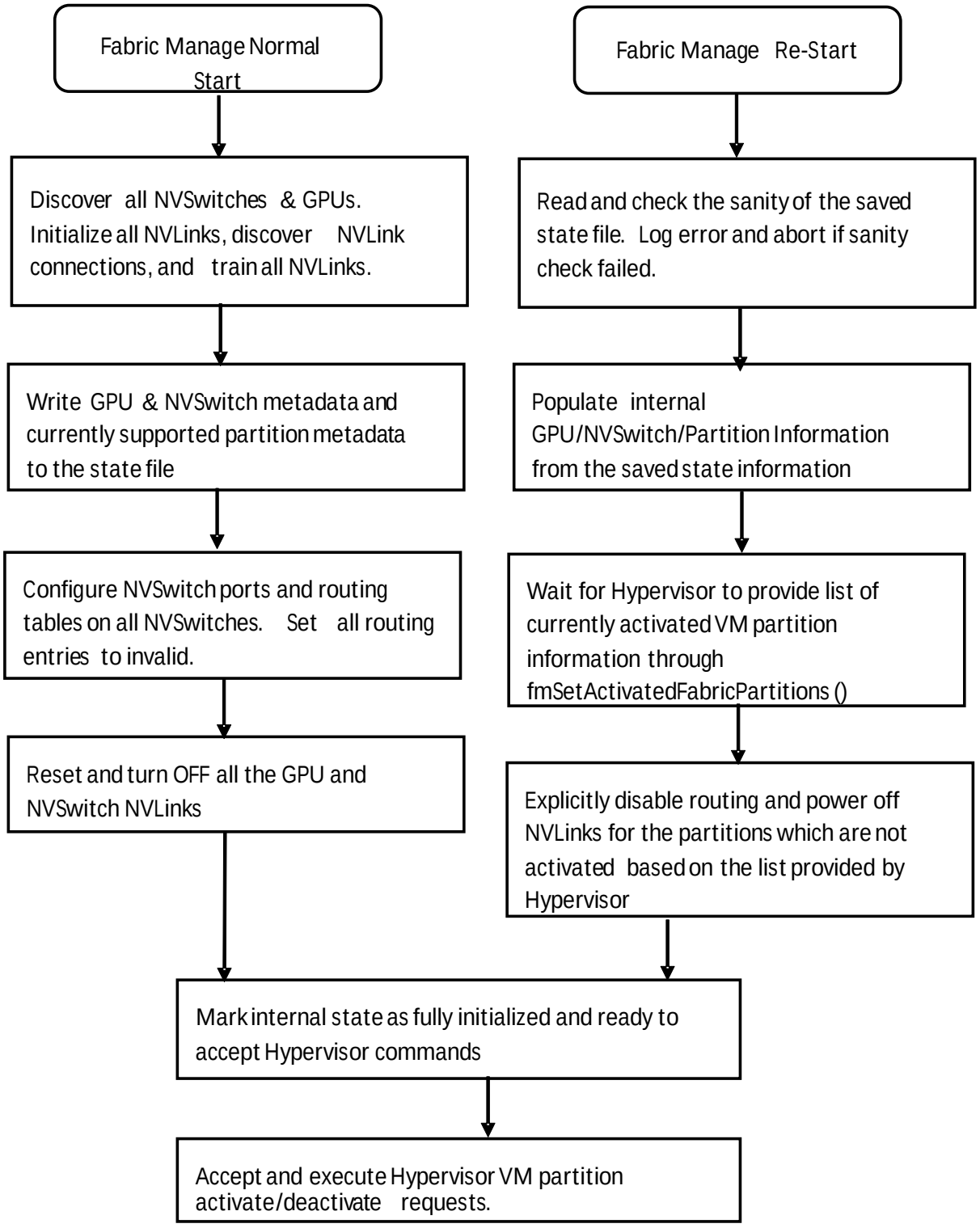
When Fabric Manager is started in normal mode, after initializing all the NVLink devices and discovering the NVLink connections, Fabric Manager will save the required metadata information into the local file system. The default file location will be /tmp/fabricmanger.state. However, this can be changed via setting the desired file location to the STATE_FILE_NAME Fabric Manager config file item. The saved state is a snapshot of detected GPU information (UUID, physical Id) and currently supported Guest VM partition metadata.

When Fabric Manager is started with the *--restar*t command line option, it will skip most of its NVLink, NVSwitch initialization steps and populate required information from the stored file. Fabric Manager will then wait for the Hypervisor to provide a list of currently activated Guest VM partitions. During this time, normal partition operations like querying the list of supported Guest VM partitions, activating and deactivating Guest VM partitions etc. will be rejected. Once the list of active Guest VM partition information is received from Hypervisor, Fabric Manager will ensure routing is enabled for those partitions only. Once these steps are completed, Fabric Manager will enable normal Guest VM partition activation and deactivation workflow.

If Fabric Manager can't resume from the current state or the Hypervisor doesn't provide the list of currently activated Guest VM partitions before the stipulated timeout, the restart operation will be aborted, and Fabric Manager will exit.

The following diagram shows a high-level flow when Fabric Manager is started with normal command line options and with this special –restart option.

Figure 7. Fabric Manager Flow Diagrams – Normal and Restart Options



Appendix D. Error Handling

D.1 FM Initialization Errors

The following errors could occur during Fabric Manager initialization and topology discovery. These errors happen only during Host boot time (vGPU mode) or Service VM initialization (Shared NVSwitch mode) and the assumption is that no Guest VMs are running.

Table 4. Errors During FM Initialization

Error Condition	Error Impact	Recovery
Access NVLink connection (GPU to NVSwitch) training failure	Depending on the ACCESS_LINK_FAILURE_MODE configuration, Fabric Manager will either disable partitions which are using the Access NVLink failed GPU or disable the connected NVSwitch (and its peer NVSwitch) and support all partition with reduced bandwidth.	· Restart FM Service (vGPU mode) or Service VM (Shared NVSwitch mode) · If the error persists, RMA the GPU.
Trunk NVLink connection (NVSwitch to NVSwitch) training failure	Depending on the TRUNK_LINK_FAILURE_MODE configuration, Fabric Manager will either remove partitions which are using Trunk NVLinks or disable the NVSwitch (and its peer NVSwitch) and support all partitions with reduced bandwidth.	· Restart FM Service (vGPU mode) or Service VM (Shared NVSwitch mode) · If the error persists, inspect/reseat the NVLink Trunk backplane connector.
Any NVSwitch or GPU programming/configuration failures and typical software errors	Treated as fatal error and Fabric Manager service will abort. However, if the FM_STAY_RESIDENT_ON_FAILURES configuration option is set, then FM service will stay running, but partition activation/deactivation flow will not be supported.	· Restart the host and FM service (vGPU mode) or Service VM (NVSwitch mode) · If the error persists, then technical troubleshooting is required.

D.2 Partition Life Cycle Errors

Table 5 summarizes potential errors returned by FM SDK APIs while querying supported partitions or activating/deactivating VM partitions.

Table 5. VM Partition Life Cycle Errors

Return Code	Error Condition/Impact	Recovery
FM_ST_BADPARAM	Provided partition ID or other parameters to the APIs are invalid	Use only the partition IDs returned by fmGetSupportedFabricPartitions(). Make sure pointer arguments are not NULL.
FM_ST_NOT_SUPPORTED	Fabric Manager is not started with required config options	Make sure that shared fabric mode is enabled in the FM config file. If not, set the desired value and restart FMservice. Shared NVSwitch Mode: SHARED_FABRIC_MODE = 1 vGPU Mode: SHARED_FABRIC_MODE = 2
FM_ST_NOT_CONFIGURED	FM APIs are issued before Fabric Manager is fully initialized	Wait until FM service is fully initialized.
FM_ST_UNINITIALIZED	FM interface library has not been initialized	Make sure FM interface library is initialized with call to fmLibInit()
FM_ST_IN_USE	Provided partition ID is already activated or the GPUs required for the specified partition is already in use on other activated partitions.	Provide a non-activated partition ID. Also make sure that the GPUs are not in use by other activated partitions.
FM_ST_UNINITIALIZED	Provided partition ID is already deactivated.	Provide an activated partition ID
FM_ST_GENERIC_ERROR	A generic error has happened while activating/deactivating a VM partition.	Check the associated syslog for specific and detailed error information.
FM_ST_TIMEOUT	A GPU or NVSwitch configuration setting timed out.	Check the associated syslog for specific and detailed error information.
FM_ST_ VERSION_MISMATCH	The client application using the FM APIs might have compiled/linked with a different version of Fabric Manager package running on the Service VM.	Make sure the client application is compiled and linked with the same or compatible Fabric Manager package installed on Service VM.

D.3 Runtime NVSwitch Errors

NVSwitch runtime errors can be retrieved or monitored using the following mechanisms

- Through Host or Service VM syslog and Fabric Manager log file as SXid errors
- Through the NVSwitch public API interface
- Through the NVSwitch SMBPBI based OOB commands

When a NVSwitch port generates runtime error (an SXid error), the corresponding error information and affected GPU partition information will be logged into the Host or Service VM syslog.

Depending on the type of SXid errors and the impacted port, the GPUs on the corresponding Guest VM or all other Guest VMs may be impacted. In general, if the impact is local to a Guest VM, then all the other running Guest VMs will not be affected and should function normally.

D.4 Non-Fatal NVSwitch SXid Errors

The following table lists potential NVSwitch non-fatal SXid errors which could occur in the field and their impact.

Table 6. Potential Non-Fatal NVSwitch SXid Errors

SXid & Error String	Guest VM Impact	Guest VM Recovery	Other Guest VM Impact
11004 (Ingress invalid ACL) Note: This SXid can happen only due to incorrect FM partition configuration and never expected to occur in the field	Corresponding GPU NVLink traffic will be stalled, and subsequent GPU access will hang. GPU driver on Guest VM will abort CUDA jobs with Xid 45	Validate GPU/NVSwitch fabric partition routing information using the NVSwitch-audit tool. Restart Guest VM.	If the error is observed on a Trunk port, then partitions that are using NVSwitch trunk ports will get affected
11012, 11021, 11022. 11023, 12021, 12023, 15008, 15011, 19049, 19055, 19057, 19059, 19062, 19065, 19068, 19071, 24001, 24002, 24003 (Single bit ECC errors)	No Guest VM impact as NVSwitch hardware will auto correct the ECC errors.	Not Applicable.	No Impact
20001 (TX Replay Error)	NVLink packet needs to be retransmitted. This may	Not Applicable	If the error observed on a Trunk port, then

SXid & Error String	Guest VM Impact	Guest VM Recovery	Other Guest VM Impact
	impact the NVLink throughput of the specified port.		partitions that are using NVSwitch trunk ports may see throughput impact.
12028 (egress non-posted PRIV error)	Corresponding GPU NVLink traffic will be stalled, and subsequent GPU access will hang. GPU driver on Guest VM will abort CUDA jobs with Xid 45	Restart Guest VM	If the error is observed on a Trunk port, then partitions that are using NVSwitch trunk ports will get affected
19084(AN1 Heartbeat Timeout Error)	Usually, this error is accompanied by a fatal SXid which will affect Corresponding GPU NVLink traffic	Restart Guest VM	If the error is observed on a Trunk port, then partitions that are using NVSwitch trunk ports will get affected
22013(Minion Link DLREQ interrupt	Indicates a Switch programming error and usually accompanied by other errors.	Follow the action based on the accompanied error.	This error happening on Trunk port could affect other VMs, but final action is based on the accompanied error.

D.5 Fatal NVSwitch SXid Errors

The following table lists potential NVSwitch fatal SXid errors which could occur in the field. The Hypervisor must track these SXid source ports (NVLink) to determine if the error occurred on an NVSwitch trunk port or NVSwitch access port. The fatal SXid will be propagated to the GPU as Xid 74 when applicable.

- If the error occurred on an NVSwitch access port, then the impact will be limited to the corresponding Guest VM. The Guest VM needs to be shut down to recover.
- If the error occurred on an NVSwitch trunk port, then all Guest VM partitions that are crossing the trunk port will need to be shut down to reset the trunk ports and recover. After that, the partitions can be recreated. As of now, the partitions that are using NVSwitch trunk ports are the 16x GPU partition and the 8x GPU partitions with 4 GPUs per baseboard.

Table 7. Potential Fatal NVSwitch SXid Errors

SXid	SXid Error String
11001	ingress invalid command
11009	ingress invalid VCSet

SXid	SXid Error String
11013	ingress header DBE
11018	ingress RID DBE
11019	ingress RLAN DBE
11020	ingress control parity
12001	egress crossbar overflow
12002	egress packet route
12022	egress input ECC DBE error
12024	egress output ECC DBE error
12025	egress credit overflow
12026	egress destination request ID error
12027	egress destination response ID error
12030	egress control parity error
12031	egress credit parity error
12032	egress flit type mismatch
14017	TS ATO timeout
15001	route buffer over/underflow
15006	route transdone over/underflow
15009	route GLT DBE
15010	route parity
15012	route incoming DBE
15013	route credit parity
19047	NCISOC HDR ECC DBE Error
19048	NCISOC DAT ECC DBE Error
19054	HDR RAM ECC DBE Error
19056	DAT0 RAM ECC DBE Error
19058	DAT1 RAM ECC DBE Error
19060	CREQ RAM HDR ECC DBE Error
19061	CREQ RAM DAT ECC DBE Error
19063	Response RAM HDR ECC DBE Error
19064	Response RAM DAT ECC DBE Error
19066	COM RAM HDR ECC DBE Error
19067	COM RAM DAT ECC DBE Error
19069	RSP1 RAM HDR ECC DBE Error
19070	RSP1 RAM DAT ECC DBE Error
20034	LTSSM Fault Up
22012	Minion Link NA interrupt

SXid	SXid Error String
24004	sourcetrack TCEN0 crubmstore DBE
24005	sourcetrack TCEN0 TD crubmstore DBE
24006	sourcetrack TCEN1 crubmstore DBE
24007	sourcetrack timeout error

D.6 Always Fatal NVSwitch SXid Errors

The following table lists potential NVSwitch fatal SXid errors which are always fatal to the entire fabric/system. After an always fatal SXid error has occurred, all the Guest VM partitions need to be shut down and either the host needs to be restarted or a Service VM restart required (after all the NVSwitches and GPUs are SBRed).

Table 8. Always Fatal NVSwitch SXid Errors

SXid	SXid Error String
12020	egress sequence ID error
22003	Minion Halt
22011	Minion exterror
23001	ingress SRC-VC buffer overflow
23002	ingress SRC-VC buffer underflow
23003	egress DST-VC credit overflow
23004	egress DST-VC credit underflow
23005	ingress packet burst error
23006	ingress packet sticky error
23007	possible bubbles at ingress
23008	ingress packet invalid dst error
23009	ingress packet parity error
23010	ingress SRC-VC buffer overflow
23011	ingress SRC-VC buffer underflow
23012	egress DST-VC credit overflow
23013	egress DST-VC credit underflow
23014	ingress packet burst error
23015	ingress packet sticky error
23016	possible bubbles at ingress
23017	ingress credit parity error

D.7 Other Notable NVSwitch SXid Errors

The following section details additional SXid errors that could affect the overall fabric/system.

Table 9. Other notable NVSwitch SXid Errors

SXid	SXid Error String	Comments/Description
10001	Host_priv_error	The errors themselves are not fatal to the fabric/system. But they could be followed by other fatal events
10002	Host_priv_timeout	The errors themselves are not fatal to the fabric/system. But they could be followed by other fatal events
10003	Host_unhandled_interrupt	This SXid error is never expected to occur. If it occurs, then it is fatal to the fabric/system and will require a system restart to recover
10004	Host_thermal_event_start	Related to thermal events. They are not directly fatal to the fabric/system but do indicate that system cooling may be insufficient. Also, this may force the specified NVSwitch Links to enter power saving mode (Single Lane Mode) and impact over all NVLink throughput
10005	Host_thermal_event_end	Related to thermal events. They are not directly fatal to the fabric/system, but do indicate that system cooling may be insufficient

Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation ("NVIDIA") makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer's own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer's sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer's product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

VESA DisplayPort

DisplayPort and DisplayPort Compliance Logo, DisplayPort Compliance Logo for Dual-mode Sources, and DisplayPort Compliance Logo for Active Cables are trademarks owned by the Video Electronics Standards Association in the United States and other countries.

HDMI

HDMI, the HDMI logo, and High-Definition Multimedia Interface are trademarks or registered trademarks of HDMI Licensing LLC.

OpenCL

OpenCL is a trademark of Apple Inc. used under license to the Khronos Group Inc.

Trademarks

NVIDIA, the NVIDIA logo, NVLink, and NVSwitch are trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
http://www.nvidia.com