XINNOR

Xinnor xiRAID Opus User Manual

Contents

Version	Date	Description
v.1	11-APR-2024	xiRAID Opus v 1.0.0 User Manual

Xinnor xiRAID Opus 1.0.0 User Guide

Instructions, practical guides, and tips for administering Xinnor xiRAID Opus 1.0.0

Acronyms and Definitions

API	Application programming interface
BDEV	Any logical block storage item managed by xiRAID block layer. BDEVs can be stacked and form multi-level topology.
Block device	Any backend storage block device, such as an NVMe namespace, logical BDEV, or a frontend block device exposed by a target of any type (Vhost, NVME-TCP, NVME-RDMA or other).
Chunk	The unit in which data is spread over the RAID disks. One chunk of continuous data is written to a certain disk, the next data is written to another disk.
CLI	Command Line Interface implemented by xiRAID Opus utility for the xiRAID engine management.
Device, physical device	A device directly attached via PCI or network, either physically or virtually through a hypervisor. The device can represent a single drive or multiple drives storing user and RAID control data. An example of such a device is a PCIe-connected NVMe.
DMA	A device directly attached via PCI or network, either physically or virtually through a hypervisor. The device can represent a single drive or multiple drives storing user data. An example of such a device is a PCIe-connected NVMe drive.
Drive, disk	The lowest level logical block device created on top of a physical or virtual device. It is internally represented as BDEV. It can be used as a backing disk for a RAID.

(continued)

Driver	A specific computer program enabling the operating system or an application to access to a hardware target device. Usually to a disk or network port in scope of this document subject.
gRPC	Google Remote Procedure Calls protocol is a framework based on the proto3 specification.
Host	A machine where xiRAID Opus software is installed and running.
Initiator	The endpoint that initiates a RAID data access session. the initiator usually is a part of user's application.
IO	Input/output is the flow of data to and from the storage drive, backend disk, target, or another data path component.
IOMMU group	An IOMMU group is the smallest set of devices that can be considered isolated from the IOMMU's perspective. Devices from the same IOMMU group should not be split between different users and host drivers.
InfiniBand	A computer networking communications standard used for data interconnection.
Metadata	Persisted RAID control parameters located at the RAID backing drives. the metadata is not available for client application and is used for the RAID operation and data recovery.
NIC	The Network Interface Controller enables computers to communicate over a computer network.
NVMe-oF	A specification-defined extension to NVMe that enables NVMe-based communication over interconnects other than PCIe. This interface makes it possible to connect "external" storage enclosures to a server, either directly or through a switch, while still using NVMe as the fundamental communication mechanism.
Namespace	A quantity of nonvolatile memory that can be formatted into logical blocks. This definition is virtually identical to the

(continued)

	SCSI concept of a logical unit. A logical block device that is part of physical NVMe device.
NQN	NVMe Qualified Name. An identifier for a remote NVMe target.
OS	Operating System.
RAID	Redundant Array of Independent Disks is a single usable data disk, where several physical disks are combined into an array for better speed and fault tolerance.
Server	The machine where xiRAID software is running.
SSD	Solid State Drive.
Subsystem	The NVM subsystem is a collection of physical fabric interfaces (ports), one or more controllers, one or more namespaces and an interface between the controller(s) and non-volatile memory storage medium.
Target	The endpoint that waits for initiators' commands and provides required input/output data transfers. The part of xiRAID software.
UUID	Globally unique 128-bit identifier. Unique identifier of a disk, logical drive, RAID or a configuration component
VirtIO transport	An abstraction layer in the hypervisor to expose a software block devices to the guest as if they are physical devices using a specific transport method
VM socket	The character device used to connect a block storage by using VirtIO protocol.
xiRAID	Xinnor xiRAID Opus product or engine
xiRAID Opus	xiRAID Opus (Optimized Performance in User Space) is a high-performance software RAID engine based on the SPDK libraries

Introduction

Intended Audience

This guide is intended for administrators and users of block storages based on the Xinnor xiRAID Opus software. It provides details on the xiRAID Opus operation and instructions on how to configure and manage objects in a xiRAID Opus storage.

About the product

The xiRAID Opus software implements a storage system using a user-space data path engine and management components. The user-space engine enables the system to achieve high speeds and eliminates dependencies on the operating system version, drivers, and libraries.

xiRAID Opus Components

The storage system consists of the following components:

- **Physical storage devices (SSD)** inserted into a server or connected via network. Current xiRAID Opus version supports NVMe SSD inserted into a server, with plans to support network-connected devices in future versions.
- **Backend disk drivers** that facilitate the flow of IO data between disks and the data path engine. xiRAID uses OS drivers for communication to a device at PCIe or network controller level. It uses user-space NVMe or network NVME-RDMA drivers to handle the main disk communication IO logic.
- **Device Manager** component responsible for connecting, disconnecting, and monitoring physical storage devices and their drivers within the data path engine and OS.
- **Block device (BDEV) subsystem** that represents unified logical disk API for different disk types such as a backend disk, RAID, or other block device within a layered block device topology.
- **RAID engine** based on the fast Xinnor xiRAID technology. The RAID engine implements various RAID features such as RAID structure initialization, general IO operations, degraded mode IO, disk replacement, and disk data

reconstruction (including Xinnor's partial reconstruction algorithms). Future releases will also support hot spare disks.

- **Frontend target subsystem** that connects xiRAID block devices or other BDEVs configured on top of a RAID to a client IO engine. The current release includes support for VirtIO targets, as well as experimental NVMe-TCP and NVMe-RDMA target types.
- **Configuration database** that persists and reports the storage topology and its component parameters. The database enables the automatic recovery and restart of the storage system in the event of an engine, operating system, or physical node (server) restart.
- **Management logic** for configuring and monitoring all storage components.

xiRAID Opus Structure

The xiRAID storage engine is implemented as a single binary that runs as a service. This engine exposes a gRPC management API. An additional binary implements a CLI to represent the gRPC API as a management command-line interface. The CLI engine is a thin client that does not contain any logic. Therefore, another client (such as a GUI, customer CLI, or plugin) can connect directly to the gRPC API. The gRPC syntax is described in the gRPC API specification. Any engine command available in the CLI tool can be executed by a direct gRPC call.

The CLI uses the network layer to send commands to and receive responses from the data path engine via gRPC. This means that the CLI and xiRAID engine can operate on different network nodes, and the same CLI can manage multiple engines across various network nodes. To specify the destination engine for a command, the CLI tool needs to be provided with the network address (hostname/IP and port number). By default, if the engine and CLI are installed on the same node, the localhost and pre-configured port number are used.

Figure 1: Major parts of the xiRAID

CLI

This document uses CLI command examples to explain storage management operations, the overall approach, and common use cases. It does not provide detailed descriptions of each command and its options. For a comprehensive description of all commands and options, please refer to the Xinnor xiRAID Opus CLI specification. You can obtain a brief command description by using the self-documented CLI with the -h flag.

```
xnr_cli --help
```

```
xnr_cli <command> -h
```

The CLI implements shortcuts for commands and command options. For example, "-h" is a shortcut for "--help".

Some commands and options are hide. The --honest (or -H) flag enables help for hidden commands and options. They are experimental and not designed for production usage. The hidden commands can be used for test or prototyping. The hidden commands syntax can be changed in future releases or the commands can be removed from the CLI.

```
xnr_cli --honest --help
```

```
xnr_cli -H <command> -h
```

If a CLI command option consumes a list of items the list can be input as comma separated values or as repeated options. For example, these two commands are identical:

```
xnr_cli bdev zero --bdevs 0000:06:0a.0n1,0000:06:0b.0n1,0000:06:0c.0n1
```

```
xnr_cli bdev zero --bdevs 0000:06:0a.0n1 --bdevs 0000:06:0b.0n1 --bdevs
 0000:06:0c.0n1
```

Backend devices

The xiRAID engine is a high-performance software RAID system, leveraging multiple components to optimize performance. One key component is the internal user-space high-performance hardware drivers, facilitating efficient IO operations with local or network-connected devices. By bypassing the OS drivers, xiRAID interfaces

directly with the hardware storage devices. To use a device with the storage engine, it must be detached from the operating system and attached to the xiRAID engine for visibility. Once attached, the device is managed exclusively by xiRAID and is no longer accessible in the host OS. Upon restart, xiRAID automatically re-attaches previously connected storage devices.

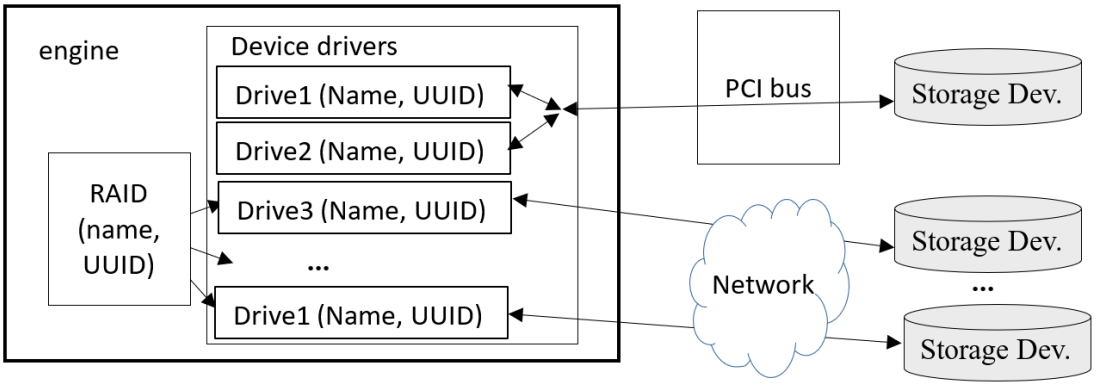

Figure 2: RAID, Drive and Storage Device Relation

Quick Start Guide

Installation

The product is delivered as ver1.0.0-b.tgz package. It should be unpacked by `tar -xvf ver1.0.0-b.tgz` at the target system. The unpacked installation package is available in the ver1.0.0-b directory as a result. The simplest way to run the installation is:

```
cd ver1.0.0-a
sudo ./install.sh -m 0xFFFF --hugemem 32768
```

The -m option specifies CPU core mask where the data path engine is running. Minimum 16 CPU cores are recommended. The --hugemem option specifies the amount of memory in Mbytes to be exclusively allocated for the xiRAID data path engine. Minimum 32GiB is recommended.

By default, the xiRAID files are installed to the /opt/xiraid/ directory. The data path engine is registered as system service and marked to autorun but not run yet. To start the engine, you should reboot the system or execute: `sudo systemctl start xnr_xiraid`

CPU core mask can specify any CPUs at any NUMA node. Another memory size is possible. It is recommended to use same NUMA node for selected CPU cores and huge memory. Detailed recommendation, installation options and full installation and uninstallation instruction is available in “Installation Procedure” section.

If you need to uninstall the engine you should run `sudo /opt/xiraid/uninstall.sh` or `sudo /opt/xiraid/uninstall.sh --force`. First uninstallation variant removes xiRAID files but keeps the engine configuration, therefore some “failed to remove” messages can be reported in this case. Second variant forces removing all files including the configuration database. Uninstallation procedure automatically stops the engine and de-registeres it from the system before removing files.

If xiRAID is uninstalled but configuration files are kept, the xiRAID can be installed back to same directory. The configuration is preserved in this case and reapplied to the engine at its next run.

By default, CLI binary is installed as a part of xiRAID package and can be run as `xnr_cli` executable. The simple command to check that data path is running is requesting the engine version by executing:

```
xnr_cli config get --name version
```

```
version : 1.0.0, build: 1096, date: 2024-02-03T12:58:44+00:00
```

Licensing

By default xiRAID comes with a trial license that has limitation on the number of drives (up to 4 in total). To remove the restrictions and work without limitations, the license needs to be installed. To get a license run the command:

```
xnr_cli license show
```

```
hwkey: 6D0D9E443400AD3E
disks
    licensed: 4
    created: 2024-03-29
    expire: 2024-04-05
    status: trial
support
    status: no
```

Then copy reported hwkey and send it to your vendor. The vendor will generate the license file specific to your hardware. After you get the license file install it by the command:

```
xnr_cli license add --path <license_file_name>
```

Licensing details are described in “License Details” section.

Connect Backend Devices

The xiRAID Opus operates using logical drives, on which any RAID configuration can be built. These logical drives become available for RAID creation only after attaching physical devices to the xiRAID Opus. Once a storage device is attached, it is managed by xiRAID and becomes inaccessible in the host operating system (see the Device Manager chapter).

Physical NVMe storage devices located at the server are discovered automatically by the xiRAID device manager. To get list of available storage devices, use the command:

```
$ xnr_cli drive-manager show --output table
```

```
┌──────────────┬────────┬──────────────────────────┬────────┐
| ID           | STATUS | PARAMETERS               | DRIVES |
├──────────────┼────────┼──────────────────────────┼────────┤
| 0000:06:0a.0 | OS     |  addr    0000:06:0a.0    |        |
|              |        |  vendor  VirtIO          |        |
|              |        |  devID   0x1001          |        |
├──────────────┼────────┼──────────────────────────┼────────┤
| 0000:06:0b.0 | OS     |  addr    0000:06:0b.0    |        |
|              |        |  vendor  VirtIO          |        |
|              |        |  devID   0x1001          |        |
├──────────────┼────────┼──────────────────────────┼────────┤
| 0000:06:0c.0 | OS     |  addr    0000:06:0c.0    |        |
|              |        |  vendor  VirtIO          |        |
|              |        |  devID   0x1001          |        |
```

Each device has ID that can be used by drive-manager commands. To attach one or several devices to xiRAID Opus, use the command:

```
xnr_cli drive-manager attach --ids
 0000:06:0a.0,0000:06:0b.0,0000:06:0c
```

The current release supports PCIe connected NVMe storage devices only.

After a storage device is attached the show command reports additional information regarding available drives. An attached NVMe device can contain one or several drives depending on the number of namespaces. Each drive has its own unique name and UUID. The name, or the UUID, or both can be used to reference the drive in a CLI command. If a drive is used by a RAID, the RAID UUID is reported by drive-manager show command.

```
$ xnr_cli drive-manager show --output table
```

```
$ xnr_cli drive-manager show --output table
┌──────────────┬────────┬──────────────────────┬──────────────────────────────────────────────────┐
│ ID           │ STATUS │ PARAMETERS           │ DRIVES                                           │
├──────────────┼────────┼──────────────────────┼──────────────────────────────────────────────────┤
│ 0000:06:0a.0 │ SPDK   │  addr    0000:06:0a.0 │ name: 0000:06:0a.0n1                             │
│              │        │  vendor  VirtIO       │ uuid: d2f67972-8fa4-4e55-8747-fe6d5ff1abbe       │
│              │        │  devID   0x1001       │ used by raid: xnraid                             │
│              │        │                       │ raidUUID: ed9753ba-a4bb-4403-a66e-e318edc7f142   │
├──────────────┼────────┼──────────────────────┼──────────────────────────────────────────────────┤
│ 0000:06:0b.0 │ OS     │  addr    0000:06:0b.0 │ name: 0000:06:0b.0n1                             │
│              │        │  vendor  VirtIO       │ uuid: 164291f3-3e07-40de-a24b-bf7521df45d3       │
│              │        │  devID   0x1001       │ used by raid: xnraid                             │
│              │        │                       │ raidUUID: ed9753ba-a4bb-4403-a66e-e318edc7f142   │
├──────────────┼────────┼──────────────────────┼──────────────────────────────────────────────────┤
│ 0000:06:0c.0 │ SPDK   │  addr    0000:06:0c.0 │ name: 0000:06:0c.0n1                             │
│              │        │  vendor  VirtIO       │ uuid: 87f8c481-3e0b-41e7-aa9c-2e51546694d2       │
│              │        │  devID   0x1001       │                                                  │
```

Create RAID

The RAID is built from the drives attached by the device manager. xiRAID engine does not allow to add used disks to a new RAID. It checks disks raw data and fails adding a disk to a RAID if the disk is not empty. To clean drives run:

```
xnr_cli bdev zero --bdevs 0000:06:0a.0n1,0000:06:0b.0n1,
0000:06:0c.0n1,0000:06:0d.0n1,0000:06:0e.0n1,0000:06:0f.0n1
```

bdev names should be obtained from the drive-manager show output.

The RAID lifecycle begins with the RAID create operation:

```
xnr_cli raid create --name xnraid --level 5
 --chunk-size 64 --block-size 4096 --bdevs
 0000:06:0a.0n1,0000:06:0b.0n1,0000:06:0c.0n1,
0000:06:0d.0n1,0000:06:0e.0n1,0000:06:0f.0n1
```

The command above creates RAID level 5 with name “xnraid” from six disks using chuck size 64 Kbytes. Supported parameter values are specified in details in the “RAID Create” chapter.

After RAID is created the engine starts RAID initialization automatically. The initialization process calculates RAID parity for all RAID stripes to keep parity in consistent state even if the stripe contains garbage in term of the RAID client application. Initialization is non-blocking operation and RAID can get data and store it, but with limited performance. If a data is being written to non-initialized RAID area corresponded stripe is initialized on-demand. On initialization completed RAID goes to fully operational state. It is recommended to wait initialization complete before using the RAID for production IO.

Vhost Target

xiRAID Opus supports VirtIO Block frontend via Virtual host target connectivity. A RAID can be exposed as VirtIO block device by a command:

```
xnr_cli vhost create --ctrlr vhost.1 --bdev xnraid --cpumask 0x3
```

Where --ctrlr defines a vhost controller name to be created, --bdev option specifies the name of the RAID to be connected to, and --cpumasks specifies CPU cores to be used to process IO load for this specific vhost target. The cpumask shall be subset of the engine CPU core mask defined during the installation stage or assigned by the engine configuration command. Different vhosts can use different, the same or intersecting masks.

To get the list of all configured vhosts, use the command:

```
$ xnr_cli vhost show
[
{
  "controller": "vhost.1",
  "socket": "/opt/xraid/bin/xnr_conf/sock/vhost.1",
  "cpu_mask": "0x3",
  "iops_threshold": 60000,
  "BackendSpecific": {
    "Block": {
      "id": {
        "name": "xnraid"
      }
    }
  }
}
]
```

The “socket” parameter specifies the name to be used by a virtual machine or other client to connect the RAID as a block device.

NVME-TCP and NVME-RDMA frontend connectivities are supported in experimental mode. These target types are fully functional but their configuration is not persisted and is lost upon the engine or the server node restart.

Device Manager (DM)

In xiRAID Opus topology any RAID is built from the logical drives. The logical drives become available for RAID create operation only after physical devices are attached to the xiRAID Opus. The attach and detach operations are managed by the device manager (DM).

The DM provides the following major functions:

1. Discover available physical devices and show the list of them.
2. Attach physical devices to the xiRAID Opus so that the logical drives are ready to be used in the RAIDs.
3. Provide a list of the logical drives available in the xiRAID Opus and show which physical device contains what logical drives.
4. Detach the logical drives so that the physical device which carries them can be reused in any other application.
5. Re-attach previously attached devices after xiRAID Opus or host OS restart.

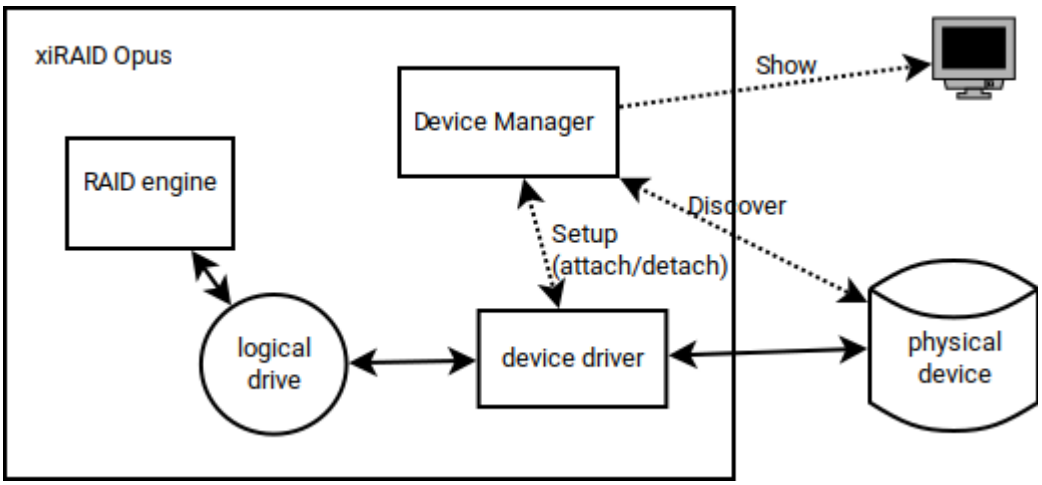

Figure 3: Device Manager

In the current release only the PCIe directly attached physical devices are supported as devices managed by the DM.

Network connected NVMe devices (by using NVME-RDMA transport) are supported in experimental mode. IO to these devices is unstable due to some problems in NVMe driver, and the network connected devices configuration is not persisted over the engine restart. `xnr_cli bdev attach` command can be used to attach network storage device to the engine. Virtual block devices provided by virtual machine hypervisor can be attached by the Device Manager if xiRAID engine is running in the virtual machine. Virtual block devices IO is unstable as well and can be used in experimental mode only.

Discover Devices

The DM discovers available devices on the host OS upon start of the xiRAID engine and periodically throughout the engine's lifecycle. Each discovered device is associated with a unique ID. The devices managed by the DM are a subset of all the

physical devices available in the host OS. DM does not report devices of unsupported types. During the discovery phase, the DM only detects the availability of devices and does not have information about their content and other parameters.

To get the list of discovered devices use:

```
xnr_cli drive-manager show --output <table,json>
```

```
xnr_cli drive-manager show --ids <device_id> --output <table,json>
```

```
┌──────────────┬────────┬──────────────────────┬──────────────────────────────────────────────────┐
| ID           | STATUS | PARAMETERS           | DRIVES                                           |
├──────────────┼────────┼──────────────────────┼──────────────────────────────────────────────────┤
| 0000:02:00.0 | SPDK   | addr   0000:02:00.0  | name: 0000:02:00.0n1                             |
|              |        | vendor WD            | uuid: 0a000000-0000-0000-0014-ee81000af490       |
|              |        | devID  0x2500        | sn: A068F6C0                                     |
|              |        |                      | name: 0000:02:00.0n2                             |
|              |        |                      | uuid: 0b000000-0000-0000-0014-ee81000af491       |
|              |        |                      | sn: A068F6C0                                     |
|              |        |                      | used by raid: test_raid                          |
|              |        |                      | raidUUID: c361637a-4fe9-4e17-96e0-4112011477ee   |
├──────────────┼────────┼──────────────────────┼──────────────────────────────────────────────────┤
| 0000:03:00.0 | USpace | addr   0000:03:00.0  |                                                  |
|              |        | vendor WD            |                                                  |
|              |        | devID  0x2500        |                                                  |
├──────────────┼────────┼──────────────────────┼──────────────────────────────────────────────────┤
| 0000:04:00.0 | USpace | addr   0000:04:00.0  |                                                  |
|              |        | vendor WD            |                                                  |
|              |        | devID  0x2500        |                                                  |
└──────────────┴────────┴──────────────────────┴──────────────────────────────────────────────────┘
```

0000:05:00.0	OS	addr 0000:05:00.0 vendor 0x15b7 devID 0x5009	
0000:06:00.0	OS	addr 0000:06:00.0 vendor 0x15b7 devID 0x5009	
0000:41:00.0	SPDK	addr 0000:41:00.0 vendor WD devID 0x2500	name: 0000:41:00.0n1 uuid: 08000000-0000-0000-0014-ee81000adefd sn: A066508B name: 0000:41:00.0n2 uuid: 09000000-0000-0000-0014-ee81000adefe sn: A066508B
0000:42:00.0	SPDK	addr 0000:42:00.0 vendor WD devID 0x2500	name: 0000:42:00.0n1 uuid: 09000000-0000-0000-0014-ee81000af487 sn: A068F61B name: 0000:42:00.0n2 uuid: 0a000000-0000-0000-0014-ee81000af488 sn: A068F61B
0000:43:00.0	SPDK	addr 0000:43:00.0 vendor WD devID 0x2500	name: 0000:43:00.0n1 uuid: 09000000-0000-0000-0014-ee81000af307 sn: A068F54B used by raid: test_raid raidUUID: c361637a-4fe9-4e17-96e0-4112011477ee name: 0000:43:00.0n2 uuid: 0a000000-0000-0000-0014-ee81000af308 sn: A068F54B used by raid: test_raid raidUUID: c361637a-4fe9-4e17-96e0-4112011477ee

The output of the show command is represented as table or json format depending on the `--output` option. Each line of table or item in the json array represents particular physical device and provides the following information:

- The unique ID of the device assigned by the DM. The ID is used to reference the device in DM commands.
- The device parameters such as PCI address, vendor name and hardware model identifier
- The device status: OS, US, SPDK, or LOST
- If the device is attached, name, UUID and serial number of the logical drive(s) hosted by this device
- If a drive of the device is used by a RAID, the RAID name and UUID

In the example above the devices `0000:05:00.0` and `0000:06:00.0` are managed by OS, the devices `0000:41:00.0` and `0000:42:00.0` are managed by xiRAID engine and contain two logical drives each. the device `0000:02:00.0` contains two logical drives and the second drive is used by the RAID with name `test_raid` and UUID `c361637a-4fe9-4e17-96e0-4112011477ee`.

Device State Machine

The device status reports the state in the device lifecycle state machine.

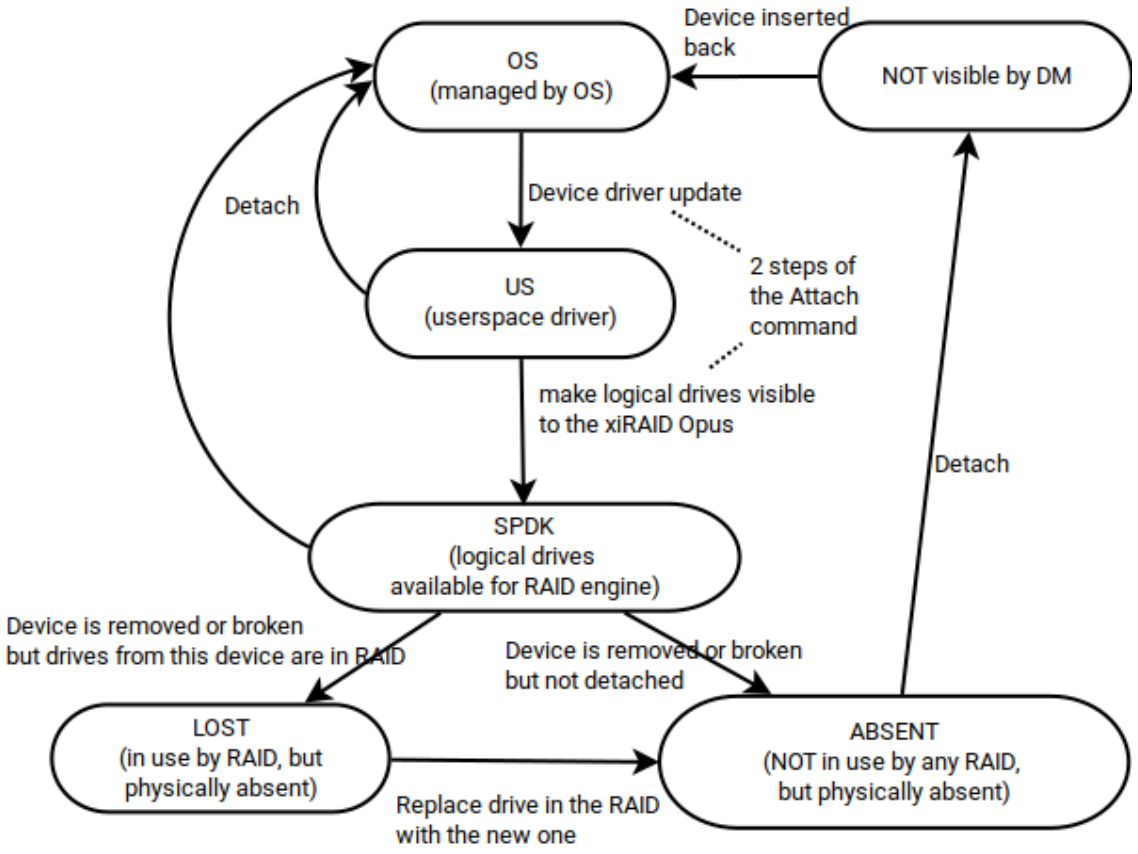

Figure 4: Device life cycle state machine

The possible states are:

- **OS** – The device is managed by host OS. The OS drivers are used for IO to the device's drives. The device cannot be used by a RAID or other xiRAID Opus modules.
- **USpace** – The device driver has been updated, and the host OS does not manage it. However, the logical drives located on this device are not attached as BDEVs and are still not available as RAID backing drives. This is an intermediate state between fully attached and detached cases. Usually the state means that the device is detached but the xiRAID engine could not restore OS drivers. Attach or Detach command should be used to move the device to SPDK or OS state respectively.
- **SPDK** – The device driver has been updated, and the logical drives located on this device are attached as BDEVs. They can be used as backing drives for a RAID.
- **LOST** – The logical drive located on the device is used in a RAID configuration, and the physical device has been removed or is malfunctioning. The corresponding RAID state is reported as Degraded. To recover the drives and reconnect them back to the RAID the physical device has to be inserted back, the device should be re-attached by DM Attach command, then the drive should be inserted to the corresponded RAID logical position by the "raid replace" command.
- **ABSENT** – Similar to LOST, but the drives of the device are not used in any RAID. The DM automatically stops to manage the device. Absent devices are not reported by the DM show command.
- **NOT VISIBLE** – The device does not exist or cannot be managed by the DM. Such devices are not reported by the DM show command.

Device BDEVs

Internally any logical drive is represented as BDEV block device. Therefore, drives of an attached device are reported in scope of BDEVs list. To get the list of all available devices attached to SPDK, use the command:

```
xnr_cli bdev show
```

Here is an example of `0000:06:0e.0n1` drive created during attaching `0000:06:0e.0` device and reported in scope of BDEVs list:

```
{
        "assigned_rate_limits": {
                "r_mbytes_per_sec": 0,
                "rw_ios_per_sec": 0,
                "rw_mbytes_per_sec": 0,
                "w_mbytes_per_sec": 0
        },
        "block_size": 512,
        "claimed": false,
        "name": "0000:06:0e.0n1",
        "num_blocks": 10485760,
        "product_name": "VirtioBlk Disk",
        "supported_io_types": {
                "abort": false,
                "compare": false,
                "compare_and_write": false,
                "flush": false,
                "nvme_admin": false,
                "nvme_io": false,
                "read": true,
                "reset": true,
                "unmap": true,
                "write": true,
                "write_zeroes": true
        },
        "uuid": "2dd51764-7ebb-4ce0-8075-286b99027c82",
        "zoned": false
},
```


Attach

To use a device's drive(s) in xiRAID engine logic the device has to be attached first. The attach operation replaces the device drivers and exposes drives hosted by the device to the xiRAID engine. After attachment, the device is no longer managed by the host OS. The drives can be used in scope of new or previously configured RAID. The DM keeps devices configuration and all attached devices are re-attached automatically in case of xiRAID engine restart.

Drives of the attached device are configured as BDEV block devices. Technically any such BDEV can be used out of a RAID scope. For example, the BDEV can be exposed through vhost target. However, such configuration is not designed for production and is not persisted over the xiRAID engine restart.

To attach the device(s) use the command:

```
xnr_cli drive-manager attach --ids device_id,[device_id,...]
```

where device_id is an ID reported by `xnr_cli drive-manager show`

If a device is being used by another host OS application (for example, if it contains partitions mounted to the operating system) the attach command will fail and an appropriate message will be displayed. User can force the attachment of such device, but this can lead to data corruption caused by changing the driver on the device used by the other application:

```
xnr_cli drive-manager attach --force --ids device_id
```

Detach

If the device is not needed anymore it can be detached, returned to the OS and used by other applications. The detach operation is waiting while in-progress IO to the device drives is not completed, then removes the device drive's BDEV, then the DM restores OS device driver and return the device and its logical drives to the host OS. After detach the logical drives located on the device are no longer available to any RAID.

In some cases described in the “Important Notice on device driver management” section below, the device driver is not changed back to OS and stay in US state for some time.

To detach a device, use the command:

```
xnr_cli device-manager detach --ids device_id
```

Re-attach

However, if a device is physically removed and reinserted it shall be re-attached manually to continue using the device’s drives in RAIDs.

On host OS reboot the Device Manager restores all device drivers to states used before reboot. So, all devices are reattached automatically and corresponding drives become visible as BDEVs. Users do not need to attach devices again after each reboot/restart if the devices were attached before.

If a device is physically removed from the system its state is changed to LOST. If the same or another device is reinserted to the same physical slot the device is discovered by OS and is not re-attached to the xiRAID Opus automatically. Therefore such removed and reinserted device shall be re-attached by the “device-manager attach” command.

Important Notice on Device Driver Management

The xiRAID Opus user space device driver works on the top of one of the following standard OS drivers: uio_pci_generic and vfio-pci. OS usually uses other driver by default. This is the reason to change the device driver during attachment.

The vfio-pci driver is preferable and used if both uio_pci_generic and vfio-pci drivers are available on the host OS. However, the vfio-pci driver has important limitation – the driver can be changed only for all devices in the same IOMMU group simultaneously. Therefore, if a user attach one of the devices in the IOMMU group to the xiRAID Opus, the device driver for all other devices in that group will be changed as well. See the picture below for details:

Figure 5. Attach devices located in the same IOMMU group

Here, after attach command is executed for device1 the driver is updated for both device1 and device2 as they are in the same IOMMU group. However, only the drive(s) allocated on device1 can be accessed by xiRAID Opus. Device1 has SPDK status, while the status of the device2 is US. As soon as the attach is executed for device2, drive(s) allocated on the device2 also become available for the xiRAID Opus. Device2 driver is not updated again as it was already updated on previous attach step.

The same algorithm in reverse order work for detach case. If a user has several devices in the same IOMMU group then the driver will be updated to OS driver only after all devices in that group are detached. See the picture below for details:

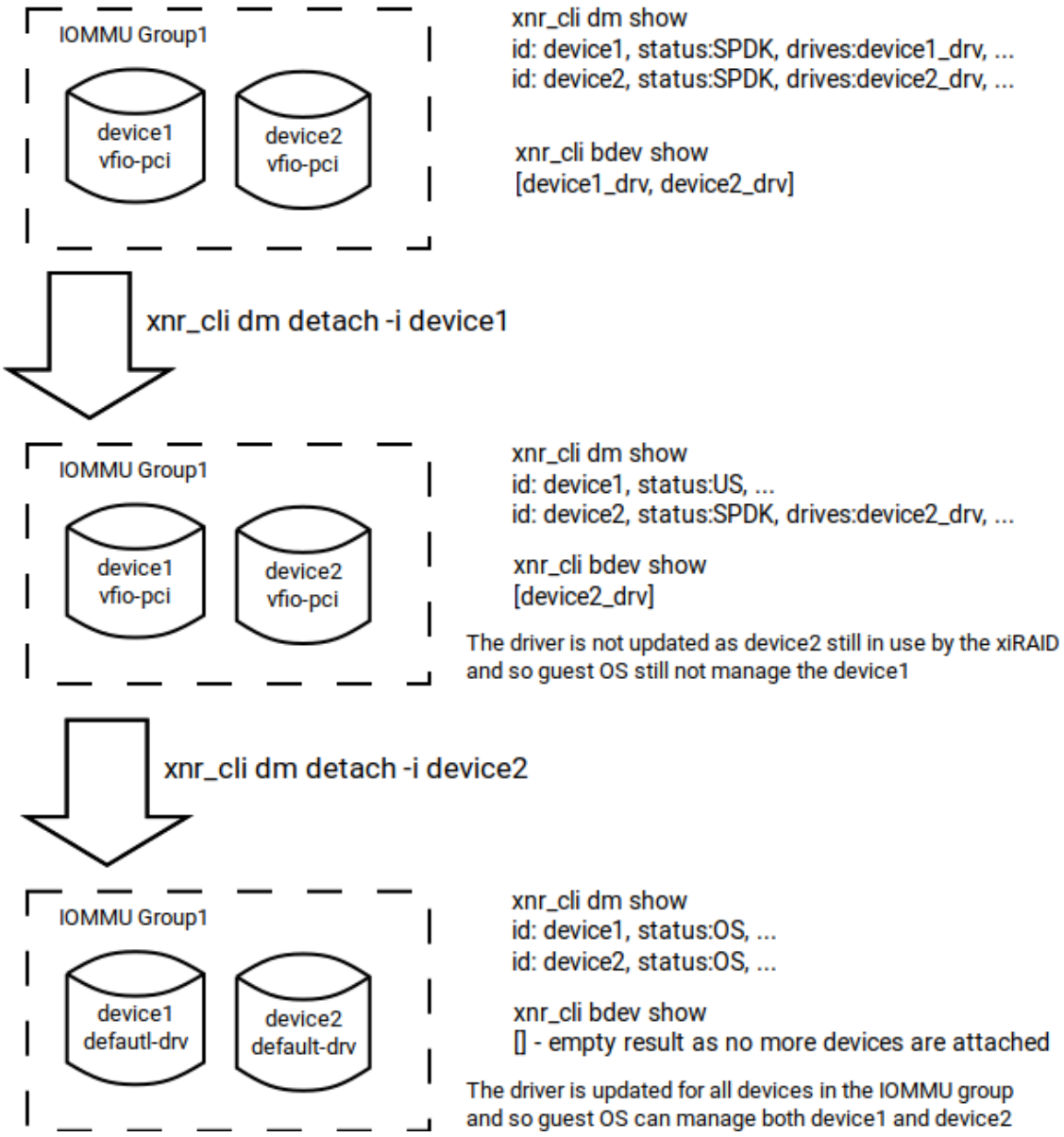

Figure 6: Detach devices located in the same IOMMU group

The allocation of devices along IOMMU groups, the number of groups and groups sizes are hardware dependent and can't be managed by the DM.

In-memory Drives (if you need a sandbox)

The product allows you to create in-memory drives that can be used to set up a RAID. However these drives has the following limitations:

1. The drives are not persisted across the xiRAID engine restart or server reboot. If a RAID is based on memory drives on restart the RAID restore will fail. To recover the situation you have to destroy such RAID manually using

"raid destroy" command, otherwise phantom RAID is kept persisted in the configuration.

2. The drives are allocated in the hugemem area, which decreases the memory available for the xiRAID engine.
3. Once created, in-memory drive can't be destroyed until the engine restart. The feature to destroy in-memory drives will be available in one of the upcoming releases.

To create an in-memory drive, use the command:

```
xnr_cli bdev malloc create -n <name> --block-size <block_size>
 --num-blocks <number_of_blocks>
```

On success the new drive will be available in the system. Its size in bytes is equal to the provided block_size multiplied to the number_of_blocks.

In-memory drives are not reported by Device Manager because they are not associated to any physical device. The in-memory drives can be listed by `xnr_cli bdev show` or `xnr_cli bdev show` -n <name> command as shown below. In-mamory drive BDEVs can be filtered by the product name equal to "Malloc disk" in the list.

Example:

```
$ xnr_cli bdev show --name memDrv0
[
{
        "assigned_rate_limits": {
              "r_mbytes_per_sec": 0,
              "rw_ios_per_sec": 0,
              "rw_mbytes_per_sec": 0,
              "w_mbytes_per_sec": 0,
        },
        "block_size": 4096,
        "claimed": false,
        "name": "memDrv0",
        "num_blocks": 65536,
        "product_name": "Malloc disk",
        "supported_io_types": {
               "abort": true,
               "compare": false,
               "compare_and_write": false,
               "flush": true,
               "nvme_admin": false,
               "nvme_io": false,
               "read": true,
               "reset": true,
               "unmap": true,
               "write": true,
               "write_zeroes": true
        },
        "uuid": "9d2eaafa-6909-4688-910e-0d13fd057a2d",
        "zoned": false
}
]
```

If there is no enough memory to create in-memory drive the error is returned:

```
$ xnr_cli bdev malloc create --block-size 4096 --num-blocks 655360
  2024-00-00T00:00:00.681Z        ERROR   Fail to call remote malloc
 create: Fail to create malloc disk, Internal error -12
```

RAID details

RAID (Redundant Array of Independent Disks) is a way of combining storage devices to ensure data integrity and high performance. There are several methods of combining hard drives called RAID levels. Each level has its pros and cons and offers a different balance of performance, data protection, and storage efficiency.

RAID levels

xiRAID Opus enables you to create RAIDs of levels 0, 1, 5, 6, 7, 10, 50, 60 and 70.

RAID 0 – interleaving data blocks without mirroring (replication of data across disks) or parity. The data blocks are distributed across several drives. The distribution results parallel IO mode that provides high performance. Due to the lack of redundancy, RAID 0 doesn't provide data reliability – the failure of one drive in RAID leads to the data corruption. RAID 0 requires at least 2 drives.

RAID 1 – mirroring without parity or striping. The data is mirrored on all drives of the RAID, and the RAID size can only be of the smallest drive size. Random read performance of a RAID 1 may equal up to the sum of each member's performance, while the write performance remains at the level of a single slowest drive. RAID 1 requires at least 2 drives.

RAID 5 – interleaving blocks with distributed parity. RAID 5 requires at least three drives. RAID 5 sustains the complete failure of one drive and provides a minimal degree of reliability.

RAID 6 – interleaving blocks with double parity distribution. RAID 6 requires at least four drives. RAID 6 can sustain the complete failure of two drives. Redundant parity information provides additional time to restore redundancy without loss of information.

RAID 7 – interleaving blocks with triple parity distribution. RAID 7 requires at least four drives. It is RAID 6 analog, but has a higher degree of reliability: three checksums are calculated using different algorithms, the capacity of three drives is allocated for checksums. Thus, the RAID 7 can sustain the complete failure of three drives.

Nested RAIDs

The architecture of RAID levels 10, 50, 60, and 70 represents RAID 0 which components are RAID 1, 5, 6 and 7 respectively instead of separate drives. These levels can be described as grouped or nested RAID levels. The nested level **group size** parameter defines how many drives are used in each of RAID 5, 6 or 7 components. RAID 10 always use group size equal to 2. the drive number in a nested RAID shall be a multiple of the group size.

RAID 10 – striped set from a series of mirrored drives that is RAID 0, which components are RAIDs 1 instead of separate drives. Each RAID 1 mirror of RAID 10 always consists of two drives, the minimum number of RAIDs 1 in a stripe array is 2. Thus, in RAID 10 the minimum number of drives is 4. Data integrity is maintained in case of failure of half drives; the irreversible RAID destruction occurs when two drives of one mirrored pair already fail.

RAID 50 – RAID 0 striping combination across multiple RAIDs level 5. With such a combination, RAID 50 may show better performance with reduced latency. The group size is at least 3 drives. Recoverable from 1 drive failure in each group.

RAID 60 – RAID 0 striping combination across multiple RAIDs level 6. RAID 60 is the equivalent of RAID 50 with a higher level of fault tolerance. The group size is at least 4 drives. Recoverable from 2 failures in each group.

RAID 70 – RAID 0 striping combination across multiple RAIDs level 7. RAID 70 is the equivalent of RAID 60 with a higher level of fault tolerance. The group size is at least 4 drives. Recoverable from 3 failures in each group.

Operational Modes

A RAID operates in different modes indicated by a set of states. Some states are interrelated, while others can be controlled independently. Changes may occur due to user actions, hardware failures, or internal logic events like initialization completion. The RAID drives have their own states. This drive state is a logical state in scope of the RAID functionality. For example, a drive can be connected to the engine and visible as a BDEV but be offline in term of RAID backing drive.

Raid show command reports the RAID states, level, size, and other useful information. It also reports the RAID backend drives and their parameters. The show output can be formatted as `--output json` or as three table formats: `table, compact, tiny`. Below is the raid show command output example:

```
$ xnr_cli raid show
```

```
┌─────────────┬──────────────────────────────────┬──────────────────────────────────────────────────────────────────────────┐
│ RAID        │ PARAMETERS                       │ DEVICES                                                                  │
├─────────────┼──────────────────────────────────┼──────────────────────────────────────────────────────────────────────────┤
│ xnraid      │ size: 9.62 GiB                   │  DEVICE                              │ PARAMETERS           │ TRANSPORT  │
│             │ level: 5                         │ ─────────────────────────────────────┼──────────────────────┼─────────── │
│ online      │ chunk size: 64 KB                │  0 0000:06:0a.0 online               │ type: VirtioBlk Disk │            │
│ degraded    │ block size: 4096 B               │  d2f67972-8fa4-4e55-8747-fe6d5ff1abbe │ size: 5.00 GiB       │            │
│ need_init   │ num blocks: 2523136              │                                      │ block size: 512 B    │            │
│             │                                  │                                      │ blocks: 10485760     │            │
│             │ service IO priority: 50%         │ ─────────────────────────────────────┼──────────────────────┼─────────── │
│             │                                  │  1 0000:06:0b.0 offline              │ type:                │            │
│             │ init progress: 47%               │  164291f3-3e07-40de-a24b-bf7521df45d3 │ size: 0.00 GiB       │            │
│             │ reconstruct progress: 0%         │                                      │ block size: 0 B      │            │
│             │                                  │                                      │ blocks: 0            │            │
│             │ UUID:                            │ ─────────────────────────────────────┼──────────────────────┼─────────── │
│             │ ed9753ba-a4bb-4403-a66e-e318edc7f142 │  2 0000:06:0c.0 online           │ type: VirtioBlk Disk │            │
│             │                                  │  87f8c481-3e0b-41e7-aa9c-2e51546694d2 │ size: 5.00 GiB       │            │
│             │                                  │                                      │ block size: 512 B    │            │
│             │                                  │                                      │ blocks: 10485760     │            │
└─────────────┴──────────────────────────────────┴──────────────────────────────────────────────────────────────────────────┘
```

RAID states

1. Online - The RAID operates normally and serves user IO.
2. Offline - If the RAID is not online the Offline state is reported. While the RAID is Offline some IO operation to its drives is possible to read or update metadata or service areas while user's IO is blocked.
3. Initializing (initing) - The parity data for the stripes is being initialized. This initialization process runs once after the RAID is created or in the event of an emergency stop and stripes resync.
4. Need Initialization (need_init) - The initialization of stripes is incomplete, and the initialization process is not running.
5. Initialized - Stripes parity areas are initialized.
6. Degraded - One or more RAID backing drives have failed. The RAID may or may not support user IO (it can be Online or Offline) depending on number of failed drives.
7. Need Reconstruction (need_recon) - A drive has failed and has been replaced with the same (reinserted and reconnected) or another drive. Data reconstruction for the drive is required.
8. Reconstructing - The RAID reconstruction is in progress.
9. Unrecoverable - If multiple drives fail and become inaccessible (cannot be reinserted or reconnected), the RAID stripes may lack sufficient parity

information to support IO and recover lost data. In such cases, manual recovery is necessary, and there is a risk of data loss or corruption.

10. Unlicensed - xiRAID engine license is expired or inapplicable to the running RAIDs. The RAID runs in read-only mode.

In addition to states listed above and reported by “raid show” command the RAID can be in the following states:

1. Not Exist - The RAID with this name, UUID and user data does not exist (was never created).
2. Stopped - The RAID IO is stopped, and the RAID configuration object is removed from the engine runtime. User data is retained on the RAID drives, and the RAID configuration is stored in the engine configuration database. User data is consistent in case of a normal RAID shutdown but can be inconsistent in case of an emergency shutdown.
3. Foreign - The RAID is stopped, and the RAID configuration is removed from the configuration database. Alternatively, the RAID is created by another RAID engine (compatible with xiRAID or incompatible with xiRAID). Such a RAID cannot run using the engine, but it retains user data. Future releases will be able to import foreign RAID if its format is compatible.

If the RAID is running and can support IO it is labeled as **Operational** in the document. If the RAID cannot support IO, it is named as **Failed**. The states of Operational and Failed are not reported by the API as they are considered integral states. Typically, a RAID is considered Operational if its state is Online. However, the RAID can be labeled as Failed if it is Online but Unlicensed.

Drive Logical States

1. Online - The device is available and operating normally.
2. Offline - The drive is currently not online and not participating in user IO. The offline device can be connected and made available for low-level IO. While

the RAID metadata can be read from or written to the offline drive, it is not recognized as being logically ready for the RAID user's IO.

3. Need Reconstruction (need_recon) - The drive data area is partially or fully incorrect and needs reconstruction from other drive data and parity. The drive can still participate in user IO operations when it requires reconstruction.
4. Reconstructing - The drive data is being reconstructed. The drive can participate in user IO while reconstructing.

RAID Lifecycle and Management

The section describes a RAID management actions and the RAID state transitions initiated by a command or some event. xiRAID engine supports many RAIDs in parallel. The RAIDs and their life cycles and states are independent each from other.

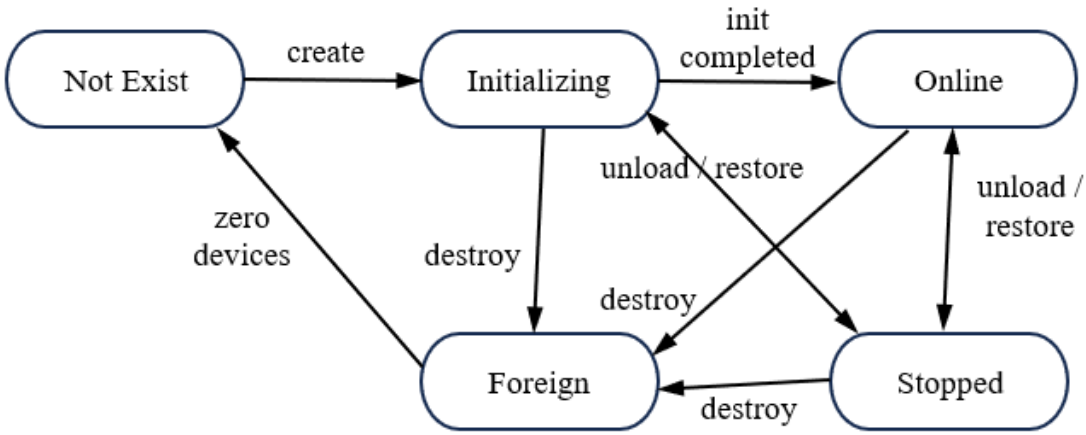

Figure 7: RAID lifecycle

After the RAID is created, it enters the initializing phase. During this phase, the drives are initialized, checksum and parity bits are built. Once the initialization process is complete, the RAID transitions to a fully operational state.

When one or more drives fail, the RAID system automatically switches to a degraded state. To restore it, the failed drive needs to be replaced. After replacing the drive, the user should start the process of reconstructing data on the new drive. During this rebuild process, performance and redundancy remain degraded until the rebuild is completed.

A RAID can be unloaded into the stopped state, where it still exists but does not perform IO and is not shown in the list of RAIDs. Drive replacement is not possible in this state.

A RAID can be destroyed and go into the foreign state from any other state.

Create

The RAID lifecycle begins from the “raid create” command. Before creating the RAID you have to make sure that the corresponded devices are attached or attach them by the Device Manager operation if needed.

The drive shall be clean to be able using them for new RAID. If first 4 Kbytes of a drive are not zero the drive cannot be used by new RAID. This prevents to add a drive with useful data by mistake. For example, a drive participated in another unloaded RAID. If drives are not clean, they can be zeroed by the following command:

```
xnr_cli bdev zero --bdevs 0000:06:0a.0n1,0000:06:0b.0n1,
0000:06:0c.0n1,0000:06:0d.0n1,0000:06:0e.0n1,0000:06:0f.0n1
```

To create a RAID use the command:

```
xnr_cli raid create --name xnraid --uuid
 75fdd4fc-d93d-484f-ae3d-cbe3f1e4e541 --level 50
 --chunk-size 64 --block-size 512 --group-size 3  --bdevs
 0000:06:0a.0n1,0000:06:0b.0n1,0000:06:0c.0n1,
0000:06:0d.0n1,0000:06:0e.0n1,0000:06:0f.0n1
```

level can be one of 0, 1, 5, 6, 7, 10, 50, 60, 70.

block-size can be 4096 or 512 bytes. It defines the RAID block device minimal IO size. If underlined drives (bdevs) use 4096 block size the RAID block size 512 is not allowed.

chunk-size specifies each drive data chunk size in Kbytes in scope of single RAID stripe. For example, RAID 5 on 8 drives with chunk size 64 has full stripe size equal to 64*7 = 448 Kbytes. Chunk size should be selected based on client IO data pattern and backend device firmware logic to optimize IO performance and latency.

uuid is an optional parameter. It is recommended to provide UUID to identify the RAID in big installations. If UUID is not provided it is autogenerated during RAID create operation and is reported by “raid show” operation. The UUID can be used together or instead of “name” parameter in any RAID related operation to identify the RAID. “name” is useful for manual input while UUID is better for scripting.

The create operation adds the RAID configuration to the persisted database and sets auto-start option to the RAID. The RAID is available for user's IO as soon as the create operation completes. If the engine is restarted or the node is rebooted the RAID is automatically restored to the operational mode (if required number of RAID's drives are attached and available). Auto-starting the RAID flow is similar to the RAID restore operation.

Initialization

After the RAID is created, the engine starts initializing drives. During the initialization process, RAID parity checksums are calculated and written to the parity area of the drives.. The RAID supports user's IO while initialization is in progress. If IO goes to uninitialized areas, the parity is being calculated on-demand. Once initialization is completed, the RAID transitions to a fully operational state.

The initialization process impacts IO performance. Without full RAID initialization, consistency checks like silent data corruption detection can't be done properly, as uninitialized areas may appear corrupted. Uninitialized areas do not impact the consistency of already written data or the reconstruct process. However, if the RAID goes into 'Degraded' mode, the reconstruct process should be run first, and initialization should be restarted after reconstruction is completed, Otherwise, reconstruct can fail to recover user's data. Because of these limitations it is recommended to wait until initialization is finished before using the RAID for user's IO.

On the raid restoration the initialization process is restarted automatically if needed. In case of emergency RAID stop (engine or the storage hardware powered down or restarted unexpectedly) the RAID is marked as possibly corrupted and the initialization process is restarted automatically to fix possible "write holes" at next RAID restore or automatic RAID restart. This process is named as "resync".

The initialization process can be stopped and restarted manually by the commands:

```
xnr_cli raid init stop --name xnraid
xnr_cli raid init start --name xnraid
```

It is possible to force finish raid initialization for performance **TEST PURPOSES ONLY** by `xnr_cli raid init finish --force --name xnraid`. This operation marks the RAID as initialized without actually calculating the stripes parity for the RAID stripes. It is strongly **NOT RECOMMENDED** to force RAID initialization as this can lead inability to

recover data and data corruption as a result if the RAID switches to degraded mode due to a drive fail.

Unload

If the RAID is not needed at the moment, you can unloaded unload it using the command:

```
xnr_cli raid unload --name xnraid
```

The unload operation switches the RAID to the Offline state, starts to reject new incoming IO requests, waits for outstanding IO completion, gracefully interrupts internal services such as initialization and reconstruction, clears the auto-start flag and stops the RAID. The Stopped RAID is not reported by the “raid show” command. Although, it is configured (persisted in the configuration database), it is not automatically restarted if the engine or the server is restarted.

In a high-availability (HA) configuration (not supported by this release), it is safe to unload the RAID at one node and restore it and start IO at another node. A RAID can be in Stopped state on multiple nodes but must not be running (Operational or Failed) at more than one node simultaneously to avoid split-brain situation and potential data corruption.

Restore

If the RAID is Stopped, it can be loaded back to Operational by the command:

```
xnr_cli raid restore --name xnraid
```

The restore operation reads the RAID configuration from persisted database, lookup the RAID drives, checks the drives' metadata, connects as many drives as possible, attempts to bring the RAID to the Online state and sets auto-start flag.

If successful, user’s IO is enabled and initialization or resync continue. Otherwise the RAID remains Offline until missing drives are attached using the "raid replace" command. Once the number of connected drives is sufficient for RAID operation, the RAID is switched to the ‘Online’ state and starts supporting user IO.

Sufficient disks number means set of available drives at appropriate logical positions to allow RAID logic read and write data considering parity areas. For example, RAID 60 can be restored if fewer than 2 drives are unavailable in each of the RAID group.

Destroy

The RAID can be stopped and removed from the engine runtime and configuration database without the possibility to restore it using the command:

```
xnr_cli raid restore --name xnraid
```

The destroy operation unloads the RAID if it is running (this step is similar to the unload operation), and then remove the RAID from the persisted configuration database. Destroyed RAID is considered as Foreign by the engine.

Destroying the RAID does not remove user data or RAID metadata from the RAID drives. A future release will be able to import Foreign RAID if its structure is compatible to the engine. If the destroyed RAID drives are going to be used by new RAID the drives metadata area should be cleaned by the “bdev zero” command or by any other way that zeroing first 4 Kbytes of the drives.

Drive Fail and Degraded Mode

A RAID supports user IO if one or more drives have failed or been disconnected. As soon as a drive cannot be used for IO, its state is changed to ‘Offline’ and the RAID state is changed to ‘Degraded’. If too many drives fail and the RAID cannot support IO, its state is changed to ‘Offline’. When the RAID goes ‘Offline’, the engine logic immediately interrupts user IO to minimize potential data damage. The RAID engine ensures data consistency except for IO packets being processed when the RAID goes ‘Offline’. These IO requests are marked as failed to the client, indicating that the data was not written. The corresponding data areas on disks may contain “old” data, “new” data or a mix of old and new data if the IO packet size is larger than a single RAID chunk size and spans multiple disks.

RAID stripes impacted by failed IO data packets may suffer from damaged parity consistency, because some stripe chunks are updated while others, including the parity area, may not be correctly updated. This scenario is referred to as a “write hole” situation.

If the RAID is marked as ‘Degraded’, follow these steps:

1. Reinsert the physical device hosting the drive. If the disconnected device is not damaged it is strongly recommended to reinsert the devices used in the RAID instead of replacing them with new ones. If the device is damaged, replace it

with a new device hosting a drive of the same or larger size. If too many logical drives are replaced by new ones, the RAID can lose too many data drives and be switched to 'Unrecoverable' state, resulting in the loss of user data.

2. Attach all devices that are reinserted by using the Device Manager. The device state can be shown using the `xnr_cli drive-manager show` command. Correctly attached drive shall report SPDK status. If a device is attached its drives are visible to the xiRAID engine as BDEVs and can be listed using the `xnr_cli bdev show` command. However, the corresponded logical RAID drives may still be Offline that means the BDEVs are not connected to the RAID logically.
3. If a drive is replaced by new one, clean it using the command `xnr_cli bdev zero --bdev <bdev_name>`, otherwise the new drive can be considered as foreign drive (used by another RAID or application) and will be declined to insert to the RAID. If the same drive used in the RAID is reinserted then do NOT zero it because it contains valid data and partial reconstruction can be applied instead of the full reconstruction algorithm.
4. Replace the Offline logical drive in the RAID with the re-inserted and re-attached drive using the command `xnr_cli raid replace --name xnraid --position 1 --bdev 0000:06:0b.0n1`. Repeat the operation for all re-attached drives. The drive state reported by the "raid show" command should change to Online.
5. If RAID is Offline (due too many drives failed and being re-inserted), unload the RAID and restore it using the commands:

   ```
   xnr_cli raid unload --name xnraid
   xnr_cli raid restore --name xnraid
   ```

6. Start the RAID reconstruction to recover data from temporary disconnected disks and address the "write hole" stripe situation. The reconstruction process also fills new disks with actual data. Use the command:

   ```
   xnr_cli raid recon start --name xnraid
   ```

When replacing drives, ensure you insert one new drive for RAID 5 (or for each group of RAID 50), two drives for RAID 6 (or each group of RAID 60), and up to three drives for RAID 7 (or each group of RAID 70). Do not replace additional drives while the reconstruction process is in progress, as this can lead to data loss.

If there is no IO in progress and disks are disconnected and the same disks are reinserted back, the RAID can switch from a Degraded (or Degraded and Offline) state back to Online without the need for reconstruction.

Replace Drive

The replace drive operation substitutes the current drive in the RAID logical position with another drive or inserts a drive to the logical position if there is no Online drive at that position.

To replace a drive in a RAID, use the command:

```
xnr_cli raid replace --name xnraid --position 1 --bdev
 0000:06:0b.0n1
```

where “position” is the drive number reported by the “raid show” command and “bdev” is the drive name reported by the “device-manager show” command.

The drive replacement logic initiated by the “raid replace” command involves the following steps:

1. If an Online drive is at the specified position, the drive is marked as Offline.
2. Wait for any outstanding IO operations to the drive to be completed.
3. Close the BDEV block device corresponds to the RAID logical drive at the specified position.
4. Open the BDEV block device specified by the 'bdev” option of the command.
5. Connect the specified BDEV as the RAID logical devices at the specified position.
6. Read the connected drive metadata to determine if the drive is brand new, a known drive at the correct position, a foreign RAID drive, or a known drive inserted at an incorrect position.
7. Update the connected drive metadata to synchronize it with the latest RAID metadata.
8. Enable the drive for user IO
9. Mark the drive as Online or Degraded depending on the data state

A new disk is always marked as Degraded. A known drive can be marked as Online if there has been no IO activity between the removal and reinsertion events (no data has changed on the drive), and marked as Degraded otherwise.

If the drive specified by “bdev” option cannot be opened or is foreign or inserted incorrectly, the replace operation will fail with the corresponding message. The previous drive will not be reinserted if the operation fails

For debugging and experimental purposes, it is possible to replace a drive with nothing, effectively removing a logical RAID drive without replacing it. Logically the operation is equal to the physical drive removal but without actual removing and detaching the physical device. However, this operation is useless in a production environment. It can be run by he following command

```
xnr_cli raid replace --name xnraid --position 1 --bdev null
```

The replace operation is needed in the following cases:

- A drive fails and the corresponding device is damaged and must be replaced by another physical device re-inserted in the same or located at a different physical slot.
- A device is unstable and needs to be proactively replaced by another physical device in the same or a different physical slot.
- A device or several devices are disconnected due to hardware fail such as a cable disconnect or disks controller failure. In this case, the disconnected disks should be reconnected (replaced with same drives used before) after the hardware problem is fixed.
- A device or several devices need to be migrated to another physical slot(s) due to a controller fail or another reason. This may involve changing PCIe address and device name. In this case, disks should be reconnected (replaced with the drive used before in the same positions).

Drive replace recommendations:

- If a device changes its PCIe address and the drive name is changed, the drive UUID has to be used to identify which drive should be reconnected to which RAID logical position. The UUID is persisted in the RAID configuration and in the drive metadata (misc. area at the disk). The UUID is reported by the “device-manager show” and by the “raid show” commands to establish a connection between a physical device’s drives (reported by device manager) and logical RAID drives (reported by the “raid show” command).
- If the connection is lost for any reason and you forget the corresponded logical slot, you can try reinserting the drive to any Offline position. The replace logic checks the UUID and blocks inserting the drive to incorrect slot (except when

inserting a new disk with zeroed metadata area). Therefore, you can safely try to reinsert the disk into each Offline position one by one until the operation is not success.

The upcoming product releases will include a feature to scan drive states and automatically reinsert temporarily disconnected drives if possible.

Device Roaming

The logical position of a RAID drive is not rigidly tied to its physical device location (slot). For example, if the RAID consists of 8 disks you can swap disks 1 and 2 between physical slots without affecting RAID's logical drive positions. Additionally, all or some disks may change their physical location (PCIe address) due to hardware reconfiguration or PCIe controller replacement. It is also possible to insert a new disk into a different physical slot than the one being replaced in the RAID array.

There are two ways to change a disk physical location. For an operational RAID the roaming process can be executed without stopping IO if the total number of simultaneously removed and reinserted disks is less than or equal to the allowed amount by the RAID level. Follow these steps:

1. Remove the disk from the slot. It is recommended to detach the disk using the device manager before removal.
2. Insert the disk into other slot and attach it using the device manager.
3. Reinsert the disks into the same logical RAID position using the "raid replace" command.
4. Run reconstruction if the reinserted disk and the RAID are marked as need reconstruction (need_recon).
5. Wait for reconstruction to complete, and then repeat the steps to move other disks.

If there is a chance to stop IO the safer way to migrate disks to another slots is:

1. Unload the RAID.
2. Detach disks by the device manager.
3. move devices between physical slots as needed.
4. Attach devices using the new slots' PCIe addresses.
5. Restore the RAID. The RAID should reconnect all drives automatically, with no need for replacement or reconstruction.

Reconstruction

If a RAID backend drive fails and replaced by the new disk or the drive is temporarily disconnected and reconnected back after some time the drive data may become invalid and shall be reconstructed from other drives data using RAID recovery algorithms.

A RAID data area is logically divided to slices. If a disk failed or removed the xiRAID Opus logic remembers what slices are modified by user IO. If same disk returns back to the RAID this information allows to identify what slices of temporary disconnected disk are impacted and shall be reconstructed and what slices data is unchanged.
If some slices are impacted only the partial recovery algorithm is used for reconstruction. The partial reconstruction logic significantly reduces reconstruction time. This is the reason always trying to reinsert temporary disconnected disk back instead of replacing it with a new disk. In case of replacing the disk with a new one all slices are marked as impacted and full reconstruction logic is used.

Many RAID levels (such as 6 and 7 or group levels 50, 60, 70) operates if more than one backend disk is disconnected. If some disconnected disks returns back (or replaced by new disks) the reconstruction can start and recover replaced disks while other disks are still disconnected.

At RAID 6 and 7 levels if several disks are removed and reinserted back one by one some slices data can be impacted at two or three reconnected disks and other slices data can be impacted at one disk only. In such case the reconstruction logic recovers most impacted slices first and less impacted slices next to minimize risk of loosing user data.

The RAID reconstruction should be initiated manually as soon as the "raid show" command reports the need for reconstruction state for the RAID. This state indicates that there are disks that have been logically inserted into the RAID and require reconstruction.

To start reconstruction, use the command:

```
xnr_cli raid recon start --name xnraid
```

The reconstruction process cannot run in parallel to initialization (resync) service. Initialization process is stopped automatically if the RAID is Degraded. If the RAID is degraded and requires initialization because the initialization process was not completed or a resync is requested, follow these steps for the correct recovery sequence:

1. Insert and attach absent devices by the device manager.
2. Re-insert (replace) all Offline logical drives by same or new drives
3. Run the reconstruction process and wait until it is completed.
4. Once the reconstruction is finished, check if the RAID has stopped reporting Degraded state.
5. Run initialization (resync) service.

The reconstruction process can be temporarily stopped, which can be useful in reducing the impact of reconstruction IO flow on the user's IO performance and latency. However, it is not recommended because it increases the risk of data loss, and the user's IO performance and latency are not optimal if the RAID is degraded.

To stop the reconstruction process, use the command:

```
xnr_cli raid recon stop --name xnraid
```

For **TEST PURPOSES ONLY**, it is possible to force finish a RAID reconstruction using the command `xnr_cli raid recon finish --force --name xnraid`. This operation marks the RAID as reconstructed (clearing Degraded and Need recon states), but it does not actually recover the data. It is important to note that this operation results in data loss and corruption. It is strongly **NOT RECOMMENDED** to force RAID reconstruction.

Resync

A RAID contains of set of stripes. Each stripe has several data chunks and one or several parity chunks. When writing data to the stripe, the RAID engine updates some data chunks and recalculate the parity chunks resulting in several IOs to the RAID backing device. So, write operation is not atomic. If a write operation is interrupted by xiRAID engine restart, a driver failure, the server OS or hardware

restart or the power down, the stripe can be logically broken and the parity chunks may not correspond to the data chunks. This scenario is known as a “write hole”.

When multiple threads are running IO operations with some IO depth, many stripes can be impacted by such interruptions. Interrupted IO operations can be completed to the client with an error or be incomplete. The RAID logic has no mechanism to recover data to “old” or “new” state and the client should perform data recovery if an IO operation is completed with an error or times out.

The RAID logic is designed to recover broken stripes' parity to restore the stripe to operational state, preparing for potential disk failures and reconstruction of the broken and replaced disk.

xiRAID Opus engine detects unexpected RAID down events and mark such RAID as dirty at next restart. The “resync” logic is implemented to locate and repair all broken stripes by recovering their parity. If a RAID is 'dirty,' the initialization restarts automatically during the next RAID restart, unless it is reconstructing or ‘Degraded’. The “resync” logic is designed as an additional initialization cycle and the RAID state reports as Initializing if the “resunc” is in progress.

Similar to regular initialization process, the resync initialization can be stopped and restarted using the commands:

```
xnr_cli raid init stop --name xnraid
xnr_cli raid init start --name xnraid
```

The resync logic can be disabled or re-enabled by setting resync_enabled configuration parameter to False or True respectively:

```
xnr_cli config set --name resync_enabled --value False
```

Current resync mode can be displayed using the command

```
xnr_cli config get --name resync_enabled
```

If resync is disabled then possible dirty RAID event will be lost. In this case the RAID can get broken stipes and this cannot be detected later.

In-memory RAID

For test purposes a RAID can be created over in-memory drives (see In-memory drives chapter). in-memory RAID shall not be used for production because in-

memory drives lose its data over the engine or the server restart. Here is the example how to create an in-memory RAID by the CLI commands:

```
$ xnr_cli bdev malloc create --block-size 4096 --num-blocks 65536
$ xnr_cli bdev malloc create --block-size 4096 --num-blocks 65536
$ xnr_cli bdev malloc create --block-size 4096 --num-blocks 65536
$ xnr_cli bdev malloc create --block-size 4096 --num-blocks 65536
$ xnr_cli raid create --name xnraid --level 5 --chunk-size 16
 --bdevs Malloc0,Malloc1,Malloc2,Malloc3
$ xnr_cli raid show --name xnraid --output tiny
```

```
┌──────────────┬──────────────────┬─────────┐
| RAID         | PARAMETERS       | DEVICES |
├──────────────┼──────────────────┼─────────┤
| xnraid       | size: 0.19 GiB   | 4       |
| online       | level: 5         |         |
| initialized  |                  |         |
|              |                  |         |
└──────────────┴──────────────────┴─────────┘
```

Frontend Connectivity

xiRAID Opus enables you to work with high-performance Vhost and NVMe-oF targets, which can expose block devices and RAIDs over the network or to multiple clients. NVMe-oF targets operates in experimental mode in this release. Therefore some limitations are applicable as described below.

Vhost Target

Each RAID created in the xiRAID Opus can be exposed to a client over VirtIO transport. After the RAID is exported, a Vhost socket is created on the FS in the `<install_dir>/bin/xnr_conf/sock` directory. This Vhost socket enables the transfer of data between a virtual machines and the RAID. A Vhost target can be created on the top of any RAID and is automatically recreated at the engine restart after the corresponding RAID is restarted.

Create Vhost Target

To create a Vhost target, use the command:

```
xnr_cli vhost create --ctrlr <vhost_name> --bdev <raid_name>
 --cpumask <value>
```

where the --ctrlr parameter specifies the name of the Vhost block target, --bdev is the name of the RAID to be exposed over this target and --cpumask is the integer number in hexadecimal format specifies which CPU cores serve this Vhost target IO. The CPU mask must be subset of the CPU mask configuration parameter specified during xiRAID Opus installation (see Installing xiRAID Opus).

The Vhost target is designed to expose a RAID block device for frontend connectivity. However, any BDEV can be exposed by Vhost target using the same “vhost create” command. If the BDEV is one of the drives managed by the Device Manager then that BDEV is re-attached at the engine restart and corresponded Vhost is also restarted. If the Vhost target is created for a BDEV that is not re-attached at the engine restart (such as malloc in-memory BDEV), then the Vhost is not restarted.

If a Vhost is configured for RAID or another BDEV that is not available anymore, for example the RAID is unloaded or destroyed the Vhost is not deleted automatically and still kept in the persisted configuration. The Vhost should be deleted if it is not longer needed.

Show Vhost Target

To show the info regarding the created Vhost targets, use the command:

```
./xnr_cli vhost show
```

or

```
./xnr_cli vhost show --ctrlr <vhost_name>
```

Here is the example output:

```
$ xnr_cli vhost show -c vh1
[
{
  "controller": "vh1",
  "socket": "/opt/xiraid/bin/xnr_conf/sock/vh1",
  "cpu_mask": "0x3",
  "delay_base_us": "0",
  "iops_threshold": "60000",
  "block": {
    "id": {
      "name": "raid1"
    }
  }
}
]
```

Delete Vhost Target

To delete a Vhost target, use

```
./xnr_cli vhost destroy --ctrlr <target_name>
```

NVMe over Fabrics Target

Current release supports NVMe over Fabric frontend connectivity as **EXPERIMENTAL** feature. NVMe-oF target configuration is NOT persisted. That means the target is not restarted if the system or xiRAID engine is restarted but the target should be recreated at any engine restart. Other limitations can be applicable to the NVMe-oF targets. NVME-oF connectivity supports RDMA and TCP transports. One of next xiRAID releases is going to support full functional NVMe-oF target configuration.

NVMe-oF TCP

TCP transport does not need any special libraries. All required software is built in scope of the xiRAID engine.

The following steps are needed to configure TCP target:

1. Create TCP transport at the xiRAID engine:

```
xnr_cli nvmf transport create --trtype TCP --zcopy
```

2. Create NVME-oF subsystem. You have to provide the subsystem nqn and serial number. Both are ASCII stings. Serial number can be any string. NQN format shall correspond to the NVMe specification. The engine validates NQN.

```
xnr_cli nvmf subsystem create --nqn nqn.2018-09.io.xinnor:node1
 --serial-number XNR00001 --model-number 'Xinnor Raid'
 --ana-reporting --allow-any-host
```

3. Add TCP listener to connect the NVMe. Specify IP address for the listener. It should be configured at a network device or you can select localhost 127.0.0.1 address. Usually localhost connectivity is slow and is suitable for test purposes. 4420 is a port number usually used for NVMe-oF.

```
xnr_cli nvmf listener add --nqn nqn.2018-09.io.xinnor:node1
 --trtype tcp --adrfam ipv4 --traddr 127.0.0.1 --trsvcid 4420
```

4. Add a RAID to the subsystem as a namespace. This operation connect the RAID block device to the subsystem as externally visible NVMe namespace through the configured transport

```
xnr_cli nvmf namespace add --nqn nqn.2018-09.io.xinnor:node1
 --bdev xnraid
```

As soon as the RAID BDEV is exposed as NVMe-oF target it can be connected by regular initiator. For example, if the listener uses localhost IP address you can connect the RAID as local Linux block device by

```
sudo modprobe nvme-fabrics nvme-tcp
sudo connect -t tcp -a 127.0.0.1 -s 4420 -n
 nqn.2018-09.io.xinnor:node1
```

Then you can list the device by nvme tool amd use the `/dev/nvme0n1` as a regualr block device for IO.

```
$ sudo nvme list
Node                  SN                   Model                                    Namespace Usage                      Format           FW Rev
--------------------- -------------------- ---------------------------------------- --------- -------------------------- ---------------- --------
/dev/nvme0n1          XNR00001             Xinnor Raid                              1          10.33  GB /  10.33  GB      4 KiB +  0 B   23.05
```

The device can be disconnected by

```
sudo nvme disconnect -n nqn.2018-09.io.xinnor:node1
```

and the namespace can be removed by

`xnr_cli nvmf namespace remove --nqn nqn.2018-09.io.xinnor:node1 --nsid 1`, where nsid is the namespace ordinal number.

There is no way to destroy TCP listener and NVMe-oF subsystem in this release.

NVMe-oF RDMA

RDMA transport requires an RDMA-capable NIC with its corresponding OFED (OpenFabrics Enterprise Distribution) software package. Before starting the NVMe-oF target with the RDMA transport, you should load the InfiniBand and RDMA modules that allow user space processes to use InfiniBand/RDMA directly. Here is a list of kernel modules that may be required: ib_cm, ib_core, ib_ucm (for older kernel versions), ib_umad, ib_uverbs, iw_cm, rdma_cm, rdma_ucm. You need to detect RDMA-compatible NICs and assign them IP addresses. You may also need to install NIC-specific kernel modules like mlx5_core, mlx5_ib.

RDMA NICs impose a limit on the number of memory regions that can be registered. The xiRAID Opus engine may begin to fail in allocating more directly accessed memory (DMA) when this limit is reached. This issue is most likely to arise when there are too many 2MB hugepages reserved at runtime. One common memory bottleneck is the number of NIC memory regions, with some NICs having a maximum limit of 2048 memory regions. This effectively sets a 4GB memory limit with 2MB hugepages for the total memory regions. Using pre-reserved 1GB hugepages memory can help overcome this limitation.

Another known issue arises when using the E810 NICs in RoCE mode. Specifically, the NVMe-oF target may encounter difficulty in destroying a qpair due to unflushed posted work requests. This can result in the NVMe-oF target being unable to terminate cleanly.

Similar to TCP the RDMA target can be configured by the following commands:

```
xnr_cli nvmf transport create --trtype RDMA
xnr_cli nvmf subsystem create --nqn nqn.2018-09.io.xinnor:node1
 --serial-number XNR00001 --model-number 'Xinnor Raid'
 --allow-any-host
xnr_cli nvmf listener add --nqn nqn.2018-09.io.xinnor:node1 --trtype
 rdma --adrfam ipv4 --traddr 10.10.10.10 --trsvcid 4420
xnr_cli nvmf namespace add --nqn nqn.2018-09.io.xinnor:node1 --bdev
 xnraid
```

Then the RAID block device can be connected by an initiator

```
sudo modprobe nvme-fabrics nvme-rdma
sudo nvme discover -t rdma -a 10.10.10.10 -s 4420
sudo nvme connect -t rdma -a 10.10.10.10 -s 4420 -n
 nqn.2018-09.io.xinnor:node1
```

The RDMA connection can require additional configuration at the target or initiator side or at the switch.

Engine Configuration

Memory Configuration

The xiRAID Opus uses huge memory pages. The amount of huge memory needs to be specified during the product configuration at installation. Failing to configure the huge memory will result in the product not starting or functioning properly. You can check the size of the configured huge memory, using the command:

```
$ xnr_cli config get --name hugemem_2mb
```

After the initial configuration and the engine, this parameter can be updated using the command:

```
$ xnr_cli config set --name hugemem_2mb --value <new_hugemem_value>
```

where new_hugemem_value is the comma separated list of hugemem memory size in megabytes to be allocated at corresponded NUMA nodes. For example, the “16000,8000” value instruct the engine to allocate 16000 MB at the first NUMA node and 8000 MB at the second NUMA node. The value “32000” instructs to allocate 32

GB at first NUMA node only and the value “0,32000“ instructs to allocate 32 GB at the second NUMA node only. The current xiRAID release does not support memory domains. For optimal performance it is recommended to allocate the memory at one NUMA node anly and the node number should corresponds to the CPU mask where the engine is running. The xiRAID service shall be restarted to apply new memory configuration.

There is no limitation on the hugemem size except for the size of the physical memory on your hardware. It is recommended to set it to at least 32 GB. If you pass too big value for the hugemem the following error will be in the syslog and the actual hugemem parameter will not be updated:

```
xnr_plt_set_hugemem: ERROR: Node 0, hugepages 2048kB requested
 450000, allocated 4578
```

CPU Mask

The xiRaid Opus utilizes more than one CPU cores. You have to specify the number of cores that the product can use. This is a mandatory parameter and must be configured during the installation. It can be passed via the -m parameter as core mask (like 0xF) or core list of '[]' embraced (like [0,1,10]). If your system has several NUMA nodes it is recommended to select CPU cores and allocate hugemem at same NUMA node for optimal performance.

```
$ sudo xnr_xiraid -m 0x3 --xnr-hugemem 9000 --xnr-nostart
```

xiRAID allows to check and change CPU mask in runtime. To check the configured CPU mask use the command:

```
$ xnr_cli config get --name cpu_mask
```

This parameter can be updated by the following commands. The xiRAID service shall be restarted to apply new CPU mask value:

```
$ xnr_cli config set --name cpu_mask --value 0xfffe
```

Other Configuration Items

xiRAID Opus system configurations can be managed using the “config” command. To get list of all the available parameters use the command:

```
xnr_cli config get
```

Here is the command output example:

```
$ xnr_cli  config get
hugemem_2mb            :  9000
resync_enabled         :  True
log_flags              :
engineering_mode       :  False
version                :  1.0.0, build: 1234, tag: ver1.0.0-b, date:
 2024-03-08T20:17:23+00:00
commit_spdk            :  07329a7cde83ca57dba97d68db49c32eeeb0fe3f
cpu_mask               :  0x3
commit_base            :  2cd31a6122e547342595c459bd395305938dcf96
log_level              :  NOTICE
log_print_level        :  NOTICE
```

Some parameters are read-only (for example commit_base and version). Other parameters can be updated by the command:

```
$ xnr_cli config set --name Name --value NewValue
```

If you try to update read only parameter you will get the error message:

```
$ xnr_cli  config set --name engineering_mode --value true
2024-00-00T18:06:09.726Z        ERROR   Fail to update the
 configuration: cannot change the parameter
```

The current release support the following parameters:

- **version** (read only) - The xiRAID Opus version.
- **commit_base** (read only) - Management engine git hash.
- **commit_spdk** (read only) - Data path engine git hash.
- **engineering_mode** - Always False in normal operation mode. Support engineer can enable the mode for test and recovery.
- **log_level** - Log level for the syslog journal
- **log_print_level** - Log level for the console output if the engine running as foreground process
- **log_flags** - List of enabled log message classes for the INFO log level
- **resync_enabled** - Enable or disable the resync functionality for all RAIDs
- **cpu_mask** - CPU mask of cores used by the data path engine for processing IO
- **hugemem_2mb** - The amount of huge memory in megabytes per NUMA node used by the data path engine for internal buffers

Logging Management

xiRAID Opus Logging parameters include log_flags, log_level and log_print_level.

The log_level and log_print_level parameters can be set to ERROR, WARNING, NOTICE, INFO. INFO log level requires enabling corresponding component log_flag.

The log_level controls logging to the syslog, log_print_level controls logging to the stdout. The log_print_level is used only when xiRAID engine is running as foreground process and is not used in normal running mode where stdout is not. The syslog can be accessed and printed using `journalctl` or a similar OS command.

The log_flags enable or disable INFO logging for different components. To see the list of all components, check the output of the `xnr_xiraid -h` command in the -L flag help message. Here is the command ouput:

```
-L, --logflag <flag>    enable log flag (all, accel, accel_ioat,
 aio, app_config, app_rpc, bdev, bdev_concat, bdev_ftl, bdev_malloc,
 bdev_null, bdev_nvme, bdev_raid, bdev_raid0, bdev_raid1, blob,
 blob_esnap, blob_rw, blobfs, blobfs_bdev, blobfs_bdev_rpc,
 blobfs_rw, ftl_core, ftl_init, gpt_parse, ioat, iscsi, json_util,
 log, log_rpc, lvol, lvol_rpc, nbd, notify_rpc, nvme, nvme_vfio,
 nvmf, nvmf_tcp, opal, rdma, reactor, rpc, rpc_client, scsi, sock,
 sock_posix, thread, trace, vbdev_delay, vbdev_gpt, vbdev_lvol,
 vbdev_opal, vbdev_passthru, vbdev_split, vbdev_zone_block,
 vfio_pci, vfio_user, vhost, vhost_blk, vhost_blk_data, vhost_ring,
 vhost_rpc, vhost_scsi, vhost_scsi_data, vhost_scsi_queue, virtio,
 virtio_blk, virtio_dev, virtio_pci, virtio_user, virtio_vfio_user,
 vmd, xnr, xnr_dev, xnr_raid_config)
```

Installation Procedure

xiRAID Opus is distributed as a package containing the installation script (install.sh) and the archive with files required for installation (xiraid.tgz). You can check the full list of installed components by referring to the filelist.txt in the destination directory (by default, /opt/xiraid/filelist. txt).

The installation process consists of 3 steps: unpacking the xiRAID Opus package, executing the installation script, and starting the xiRAID Opus engine.

Prior to the xiRAID Opus installation, make sure that your system meets the requirements described in sections “Hardware Requirements”, “Supported OS” and “Required Libraries”.

installation commands presented in this section are run only with superuser privileges. Please log in as an administrator or root to run these

Hardware Requirements

The following CPU are supported by xiRAID Opus:

- 64-bit Intel processors
- 64-bit AMD processors supporting AVX2 instructions set

Supported OS

The product test cycle is executed on the:

- Ubuntu 22.04.1 LTS.
- Rocky Linux 9.3

Since the product is delivered as pre-compiled Linux OS binaries, it can be run on any x64 Linux OS that meets the limitations described in this chapter. The application autostart is implemented using systemd. If your OS does not support it, you will need to set-up the application auto-start manually.

The CLI utility connects to the xiRAID Opus via the gRPC network interface. If CLI and the product are installed on different hosts, the connection port must be open in the firewall (by default it is 8000).

Required Libraries

xiRAID Opus requires the following packages to be pre-installed on your system:

- GLIBC_2.34 (libstdc++.so.6, libc.so.6)
- openssl3 (libssl.so.3)
- crypto (libcrypto.so.3)
- libnl-3 (libnl-route-3.so, libnl-3.so)
- libgcc_s.so.1
- linux-vdso.so.1
- libnuma.so.1
- libibverbs.so.1
- librdmacm.so.1
- libuuid.so.1
- libaio.so.1
- libm.so.6

Installing xiRAID Opus

The commands described in this chapter require superuser privileges. Please log in as an administrator to execute them. By doing so, the script will be able to create autorun files and access the destination directory for file installation. If you do not need autorun feature and has access to the destination directory then install script can be run as regular user, however the product itself must be run with the superuser privileges in any case.

Installation steps:

1. Download the xiRAID Opus package from the Xinnor website and unpack it using the command:

```
tar -xvf ver1.0.0-b.tgz
```

2. Execute the install.sh script. To do so, choose one of the following modes:

- **Install with default options** - In this mode you need to specify two extra parameters. All files will be installed to the default locations and configured using default values.

```
# ./install.sh
```

if you already know the cpumask and hugemem values you can pass it to the installation script

```
# ./install.sh --hugemem <hugemem size> -m <cpu_mask>
```

- **Interactive installation** - The script will ask regarding each installation option and wait for user input from the keyboard.

```
# ./install.sh -i
```

- **Install using command line parameters** - Each installation parameter can be defined via a command line flag listed below. If a parameter is not defined, the default value will be used.

Parameter	Description
--no-xiraid	Do not install the engine binary (the RAID engine). Only CLI tools will be installed. This is useful if you want to install the CLI on a separate workstation. The default value is to install the engine.
--no-cli	Do not install the CLI binary (the CLI tools). Only the RAID engine will be installed. The default value is to install the CLI.| |-o|The output directory where the xiraid directory, containing all the installed files, will be created. By default, it is set to "/opt".

--no-link	Do not create additional symlinks. If set this flag prevents symlink creation described in the "-l" flag section. By default this flag is not set
--no-autorun	Do not create *systemd* files to automatically run the engine at system startup. If you want to set up the engine startup manually, please use this flag. If the autorun feature is enabled, the . service file will be generated in the/ etc/systemd/system/ directory. This will enable the automatic startup of your RAID engine upon reboot. The default value is to have autorun set up.
--hugemem	Select the hugemem size (the Hugemem pages parameter) to be written to the configuration after the installation is completed
-m	Select which of the CPU cores can be used by the application (the CPU mask parameter). This mask will be written to the configuration after installation is completed
-o	The output directory where the xiraid directory, containing all the installed files, will be created. By default, it is set to "/opt".
-l	The directory where the symlinks to the installed binaries will be created. It is useful if you want to add symlinks to a directory that is included in the PATH variable. The default value is "/usr/bin".

Find below examples to install the product by using command line parameters:

```
*install to the /usr/xiraid directory and do not create symlinks.
# ./install.sh -o /usr --no-link

*install with only CLI to the /home/user/cli/xiraid directory, do
 not create symlinks and do not create systemd files to autorun the
 engine at system startup.
# ./install.sh -o /home/user/cli --no-link --no-autorun --no-xiraid
```

Testing the Installation

To ensure that the installation was successful start the RAID engine by the command:

```
systemctl start xnr_xiraid
```

Then check the engine status using the command:

```
systemctl status xnr_xiraid
```

If the output indicates that the engine is loaded and active (running), the installation has completed successfully.

```
● xnr_xiraid.service - XiRAID
     Loaded: loaded (/etc/systemd/system/xnr_xiraid.service; enabled; vendor preset: enabled)
     Active: active (running) since Thu 2023-11-16 21:31:10 MSK; 1 week 0 days ago
   Main PID: 82680 (bash)
      Tasks: 12 (limit: 19087)
     Memory: 17.0M
        CPU: 2w 2h 21min 51.544s
     CGroup: /system.slice/xnr_xiraid.service
             ├─82680 bash /opt/xiraid/scripts/run_xnr_xiraid.sh
             └─82683 /opt/xiraid/bin/xnr_xiraid.0.0.0 -m 0x3 --xnr-hugemem 8192
```

If the engine is up and running, check that the CLI is able to connect to the engine and the engine version is correct by using the command:

```
systemctl status xnr_xiraid
```

Troubleshooting

If the CLI cannot connect to the server and show the version, the reason can be the following:

1. `Error while dialing: dial tcp <port_address>: connect: connection refused.`
    - Reason 1: The appropriate port is not open and inaccessible for the CLI (by default it is 8000).
    - Reason 2: The xiRAID Opus configuration has been changed. Restart the service.
2. `*ERROR* Failed to unlink lock fd for core x, errno: 2`

    Reason: The CPU mask parameter has not been configured properly before starting the server. You can reconfigure it as described below.

3. The warning appears during xiRAID Opus installation: `*WARNING*: CONFIG: file /.../module.cfg not found`

Reason: The hugemem parameter has not been configured properly before starting the server. You can reconfigure it as described below.

4. To investigate and resolve other issues refer to the logs and use grep to search for ERROR/WARNING messages. These logs can be located in the `/var/log/syslog` file by searching for the keyword "spdk" in lowercase.

Change CPU Mask or Hugemem Size

If you specify incorrect hugemem size -the size that is bigger that can be allocated on the system then the product can't start properly. In case of incorrect hugemem the following error is displayed in the syslog:

```
xnr_plt_set_hugemem: ERROR: Node 0, hugepages 2048kB requested
 450000, allocated 4578
```

If the product is installed with wrong CPU mask or hugemem parameters and cannot start as a result you can change these parameters by calling the xiRAID Opus binary with other CPU mask and hugemem values. The operation shall be run with superuser (root) rights. --xnr-nostart flag should be used. Here is an example to update the CPU mask to 0xFFFF and hugemem size to 32768 megabytes:

```
# <path_to_installed_binaries>/xnr_xiraid.1.0.0 -m 0xFFFF
 --xnr-hugemem 32768 --xnr-nostart
```

or if you allowed symlink creation on the installation step:

```
# xnr_xiraid -m 0xFFFF --xnr-hugemem 32768 --xnr-nostart
```

Uninstalling the xiRID Opus

The process of uninstalling xiRAID Opus involves disabling system processes and removing all installed files. To do so log in as root and use the uninstall.sh script located in the target directory (by default - /opt/xiraid/uninstall.sh) or remove all files manually.

To uninstall product using the uninstall.sh script, run:

```
<install_dir>/uninstall.sh
```

The output of this command may include ‘failed to remove’ messages because by default, the uninstall.sh script does not delete the directory containing configuration files.

If xiRAID is uninstalled but configuration files are retained, xiRAID can be reinstalled to the same directory. In this scenario, the engine configuration and RAIDs configuration are preserved and will be reapplied to the engine during its next run. This allows you to restore any RAIDs you previously created.

If you do not need to save the configuration files, run:

```
/opt/xiraid/uninstall.sh --force
```

To uninstall product manually, disable the sytemd service:

```
systemctl stop xnr_xiraid
```

After that, remove all files listed in the <install_dir>/filelist.txt using the command:

```
rm -rf /opt/xiraid/bin/xnr_conf
```

License Details

xiRAID Opus product requires a license. The following features are regulated by licensing mechanism:

- Number of Disks - The total number of disks in all currently active RAIDs at the system. Disks from unloaded RAIDs are not counted.
- Support (yes/no) - Determines if users have access to support from Xinnor and can install xiRAID Opus engine updates.
- Engineering Mode (yes/no) - Grants access to special features. The mode is primarily intended for the support team rather than general customer use.

Trial License

xiRAID Opus comes with a trial license. To unlock all features and remove this restriction, users must install a personal license. If no license is installed, the xiRAID engine supports up to 4 disks for RAIDs. The trial license expiration date is set to one week after the engine startup day. This means the trial RAID goes into the “Unlicensed” state if the engine runs uninterrupted for more than a week. To re-apply the trial license, stop and re-start the engine:

```
systemctl stop xnr_xiraid
systemctl start xnr_xiraid
```

License Installation

To generate a license file, you will need to provide your hardware ID to your vendor. Once you have done so, the vendor will send you the license file specific to your hardware. To obtain your hardware ID, use the command:

```
$xnr_cli license show
hwkey: 6D0D9E443400AD3E
disks
    licensed: 4
    created: 2024-03-11
    expire: 2024-03-18
    in_use: 0
    status: trial
support
    status: no
```

Then use the reported `hwkey: 6D0D9E443400AD3E` to request a license.

The licenses are distributed as a text file. For example:

```
hwkey: 6D0D9E443400AD3E
license_key: BE304B5892B395BC60AE6AA7DD54DD2467F6179FE0F91575F9154F
8E7F3317EFEAD499B094D8B79794E5E72FF9A582312A8547824ACFD75FE017CDC24
2CC489E182F0B7BE71F5537F11F68E54FB79D7AAFF635209724A5AB9167AC23189D
7E32013EC41157150A430CF35D9AC574288AD6B898AABF4CC44F1763A38841E8AB51
version: 2
created: 2024-02-03
disks: 10
    expire: 2024-03-31
support:
    expire: 2024-03-31
```

There are two ways to install the license

- Copy license file to some directory at a system where xnr_cli is used. Then run

```
xnr_cli license add --path ./<lic_filename>
```

- Run

```
xnr_cli license add --key <license_key>
```

  where <license_key> is 256 symbols license_key string from the license file (w/o any prefixes, postfixes, white spaces or carriage return). For example:

```
xnr_cli license add --key BE304B5892B395BC60AE6AA7DD54DD2467F6179FE0
F91575F9154F8E7F3317EFEAD499B094D8B79794E5E72FF9A582312A8547824ACFD7
5FE017CDC242CC489E182F0B7BE71F5537F11F68E54FB79D7AAFF635209724A5AB91
67AC23189D7E32013EC41157150A430CF35D9AC574288AD6B898AABF4CC44F1763A3
8841E8AB51
```

Merging Licenses

If several features (like disks and support) are licensed by same license, then the expiration date is applied to all of them. However, Support feature always use its own support expiration date.

Multiple licenses can be applied to one system, with each license stored separately in the system configuration file. All installed licenses are applied sequentially at system startup. When multiple licenses are applied to one system, the features covered by these licenses are combined according to the merging rules:

1. If the license applied 0 disks, the number of available disks remains the same as the value specified in the previously applied license.
2. If the current license in use is a trial and another license is applied, the number of supported disks will be determined by the new license (if it is not 0).
3. If the number of disks in the new license is greater than in the current (non-trial) AND the expiration date of the new license is later that the expiration date

of the current license, then the new disk configuration is applied. Otherwise, the number of disks value of the new license is ignored

4. If a new disk value is applied by a license, the new number of disks OVERRIDES the old value, but does not add to it.
5. If the Support feature is enabled in a new license and the license's support expiration date is later than the current license's support expiration date, or if support was not previously licensed for the system, then the engine's support expiration date is extended to match the new license's support expiration date.
6. If the same license is applied twice then second applying is ignored and nothing is changed.

Delete License

To delete all installed license, use the command:

```
xnr_cli license remove --force
```

If licenses are deleted, the system reverts to its initial unlicensed state. If some RAIDs are running and the number of used disks exceeds the trial license limit, the corresponding RAIDs are switched to read-only mode, and their state is reported as Unlicensed.

If you have deleted licenses but have backup copy of license files, you can reinstall these licenses.

Show License State

To get the list of available licenses, use the command:

```
xnr_cli license show
```

Here is the example of valid license reported by the command:

```
key: BE304B5892B395BC60AE6AA7DD54DD2467F6179FE0F91575F9154F8E7F331
7EFEAD499B094D8B79794E5E72FF9A582312A8547824ACFD75FE017CDC242CC489
E182F0B7BE71F5537F11F68E54FB79D7AAFF635209724A5AB9167AC23189D7E320
13EC41157150A430CF35D9AC574288AD6B898AABF4CC44F1763A38841E8AB51
hwkey: 6D0D9E443400AD3E
disks
    licensed: 10
    created: 2024-02-03
    expire: 2024-03-31
    in_use: 6
    status: valid
support
    created: 2024-02-03
    expire: 2024-03-31
    status: valid
```

The `in_use` field reports how many disks are actually used in all running RAIDs at the moment.

License Expiration

If a license expires, the corresponding features go into the “Unsupported state”. If some RAIDs are running and the disks license is expired, the RAIDs are switched to read-only mode, and their state is reported as “Unlicensed”. If several licenses are installed with different expiration dates, some features may be disabled while others continue to work.

The trial license is not enabled if the actual license has expired. You have to delete installed licenses to enable trial license mode

The example of expired license:

```
$xnr_cli license show
key: 863F2A5A8F77BC6C9A20945F56BE1191BAC01CAEC366B661FE6A3974ED02
7F60BB9D38E5E30EA055EB13A033B652858C0360F4E491690100CF6B5236318AB
8F03029A1A8F69741AAC60161C754B9598870BCB9967D604D1AFCAA6B8ED309B3
608FF55305DEB09F2057C23351051DDF26DD768B7D01AD4B4E73996D903892BC61
hwkey: 6D0D9E443400AD3E
disks
    licensed: 10
    created: 2023-12-31
    expire: 2024-01-31
    in_use: 0
    status: expired
support
    created: 2023-12-31
    expire: 2024-01-15
    status: expired
```