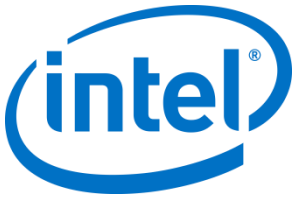

Intel® Virtual RAID on CPU (Intel® VROC)

Linux Software User Guide

June 2020

Document Number: 333915-003US

Revision History

External Version	Internal Version	Description	Date
001	0.1	Initial release	March 2016
-	1.2	VROC/RSTe 5.0 release	March 2017
-	1.3	VROC/RSTe 5.1 release	April 2017
-	1.7	VROC/RSTe 5.3 release	October 2017
-	1.10	VROC/RSTe 5.4 release	March 2018
-	2.0	VROC 6.0 release	December 2018
002	2.3	VROC 6.0 updates	November 2019
003	2.4	VROC 7.0 Updates	June 2020

Contents

1 Introduction

The purpose of this document is to help enable a user to properly set up, configure, and manage Intel® Virtual RAID on CPU (Intel® VROC) RAID volumes on NVMe drives managed by the Intel Volume Management Device (Intel® VMD) controller as well as Intel® VROC RAID volumes on SATA drives attached to the SATA and/or sSATA controllers for Linux Operating System. Within the Linux OS, the primary configuration software to manage Intel VROC RAID is the mdadm application, a native Linux tool that is used exclusively with Intel VROC on Linux.

This document describes how to use this application and many of the options offered to setup, manage and monitor Intel VROC RAID.

> ***Note:*** The information in this document is only relevant on systems with a supported Intel chipset, with a supported operating system. Please ensure that your system is properly configured to support Intel VROC.
>
> ***Note:*** The majority of the information in this document is related software configuration. Intel is not responsible for the software written by third party vendors or the implementation of Intel components in the products of third party manufacturers.
>
> ***Note:*** This user guide is not an mdadm user guide, it will only focus on those commands that are used to support Intel VROC.

Customers should always contact the place of purchase or system/software manufacturer with support questions about their specific hardware or software configuration.

The Intel VROC provides enterprise RAID support for both NVMe SSDs and SATA devices for enterprise servers, workstations and some high-end desktops.

1. Intel VROC (VMD NVMe RAID) – This product provides an enterprise RAID solution on platforms that support the Intel VMD technology.
2. Intel VROC (SATA RAID) – This product provides an enterprise RAID solution for SATA devices connected to SATA/sSATA the Intel Platform Control Hub (PCH) configured for RAID mode.
3. Intel VROC (NonVMD NVMe RAID) – This product provides an enterprise RAID solution for Intel NVMe SSDs attached to PCIe slots managed by the Platform CPU. Intel VROC (NonVMD NVMe RAID) is intended to support:
    a. Intel NVMe SSDs.
    b. Platforms that have Intel CPUs not containing Intel VMD technology.

1.1 Supported RAID volume

Intel VROC 7.0 for Linux supports the following RAID levels:

- RAID 0
- RAID 1
- RAID 5
- RAID 10
- Intel® Matrix RAID

1.1.1 RAID 0 (Striping)

RAID 0 uses the read/write capabilities of two or more drives working in parallel to maximize the storage performance of a computer system.

The following table provides an overview of the advantages, the level of fault-tolerance provided, and the typical usage of RAID 0.

Table 1: RAID 0 Overview

Drives Supported	2 minimum
Advantage	High transfer rates
Fault-tolerance	None – If one drive fails all data will be lost
Application	Typically used in desktops and workstations for maximum performance for temporary data and high I/O rate. 2-drive RAID 0 available in specific mobile configurations. It also should be noted that although RAID 0 can be scaled to many drives there is a performance sweet spot specific to your implementation.

1.1.2 RAID 1 (Mirroring)

RAID 1 arrays contain two drives where the data copied to both of the drives in real time to provide good data reliability in the case of a single disk failure. When one disk drive fails, all data is immediately available on the other without any impact to the integrity of the data.

The following table provides an overview of the advantages, the level of fault-tolerance provided, and the typical usage of RAID 1.

Table 2: RAID 1 Overview

Drives Supported	2 maximum
Advantage	Redundancy of data. One drive may fail, but data will continue to be accessible. A rebuild to a new drive is recommended to maintain data redundancy.
Fault-tolerance	Excellent – Drive mirroring means that all data on one drive is duplicated on another drive.
Application	Typically used for smaller systems where capacity of one disk is sufficient and for any application(s) requiring very high availability. Available in specific mobile configurations.

1.1.3 RAID 5 (Striping with Parity)

RAID 5 arrays contain three (minimum) or more drives where the data and parity are striped across all the drives in the array. Parity is a mathematical method for recreating data that was lost from a single drive, which increases fault-tolerance. If there are N drives in the RAID 5 array, the capacity for data would be N - 1 drives. For example, if the RAID 5 array has 5 drives, the data capacity for this RAID array consists of 4 drives.

The following table provides an overview of the advantages, the level of fault-tolerance provided, and the typical usage of RAID 5.

Table 3: RAID 5 Overview

Drives Supported	3 minimum
Advantage	High percentage of usable capacity and high read performance as well as fault-tolerance.
Fault-tolerance	Excellent - Parity information allows data to be rebuilt after replacing a failed drive with a new drive.
Application	Storage of large amounts of critical data. Not available in mobile configurations. As with RAID 0 Striping, although RAID 5 can be scaled to many drives there is a performance sweet spot specific to your implementation.

1.1.4 RAID 10

A RAID 10 array uses four drives to create a combination of RAID levels 0 and 1. It is a striped set whose members are each a mirrored set. It provides a great balance between performance and excellent fault tolerance as it allows 2 drives to fail while still maintaining access to data but, has a low cost effectiveness.

The following table provides an overview of the advantages, the level of fault-tolerance provided, and the typical usage of RAID 10.

Table 4: RAID 10 Overview

Drives Supported	4
Advantage	Combines the read performance of RAID 0 with the fault-tolerance of RAID 1.
Fault-tolerance	Excellent – Drive mirroring means that all data on one drive is duplicated on another drive.
Application	High-performance applications requiring data protection, such as video editing.

1.1.5 Intel Matrix Storage Manager

The Intel® VROC package provides high-performance NVMe RAID capabilities and allows you to create RAID arrays and spanned volumes. Matrix arrays indicate there are up to two RAID volumes in a single RAID container.

As an example, a system with an Intel® C620-A chipset, IMSM allows you to create both a RAID 0 volume as well as a RAID 5 volume across four drives. An important requirement for Matrix RAID is that the volumes within the container span the same set of member drives. When creating a container with the IMSM metadata, the mdadm "-e imsm" option is used.

Another feature of the IMSM metadata is the ability to roam an array between Linux and Microsoft Windows* host systems.

1.2 Intel® VROC RAID Write Hole Closure

The Intel® VROC product family will support the ability to close the RAID Write Hole scenario in RAID 5 configurations. This applies to Intel® VROC on Intel® Xeon® Scalable Platforms.

RAID Write Hole (RWH) is a fault scenario, related to parity based RAID. It occurs when a power-failure/crash and a drive-failure (e.g., strip write or complete drive crash) occur at the same time or very close to each other. Unfortunately, these system crashes and disk failures are correlated events. This can lead to silent data corruption

or irrecoverable data due to lack of atomicity of write operations across member disks in parity based RAID. Due to the lack of atomicity, the parity of an active stripe during a power-fail may be incorrect and inconsistent with the rest of the strip data; data on such inconsistent stripes does not have the desired protection, and worse, can lead to incorrect corrections (silent data errors).

The previous Intel® VROC mechanisms implemented to address the RAID Write Hole condition encompassed a combination of Dirty Stripe Journaling and Partial Parity Logging. This implementation only partially closed the RAID Write Hole. With Intel® VROC product family, the RWH solution included will completely close this condition (when RWH closure mechanism is enabled).

1.3 Terminology

Table 5: Terminology

Term	Description
API	Application Programming Interface
BIOS	Basic Input/Output System
Array	This term is representative of an mdadm container required for Intel metadata based volumes using the IMSM option during Volume creation.
Container	A container is a type of array used with Intel® metadata or other non-native metadata.
GB	Gigabyte
GiB	Gibibyte (1024 x 1024 x 1024 bytes)
GA	General Access – Operating System release package fully validated by Red Hat Linux.
HII	Human Interface Infrastructure
Hot-Plug	The unannounced removal and insertion of a drive while the system is powered on.
I/O	Input/Output
Initramfs	Initial Ram File system
IMSM	Intel® Matrix Software Manager
KB	Kilobyte
KiB	Kibibyte (1024 bytes)
Left-Symmetric	Default layout scheme for RAID 5 configurations. Not supported with IMSM metadata
Left-Asymmetric	Parity bit layout scheme used in RAID 5 configurations.
Matrix RAID	Two different RAID volumes within a single RAID array container.
MB	Megabyte
MiB	Mebibyte (1024 x 1024 bytes)
Member	A SATA or NVMe drive used within a RAID array.
mdadm	mdadm is a Linux utility developed to manage software RAID devices on Linux. It is available under the GPL license version 2 or later and supports SATA and NVMe SSDs.
NVMe	Non-volatile Memory Express
OS	Operating System
OSV	Operating System Vendor eg: Redhat, Suse
Pre-OS	A BIOS component to configure Intel VROC and Intel® RSTe/VROC RAID.

Term	Description
RAID	Redundant Array of Independent Disks: allows data to be distributed across multiple drives to provide data redundancy or to enhance data storage performance.
RAID 0 (striping)	The data in the RAID volume is striped across the array's members. Striping divides data into units and distributes those units across the members without creating data redundancy, but improving read/write performance.
RAID 1 (mirroring)	The data in the RAID volume is mirrored across the RAID array's members. Mirroring is the term used to describe the key feature of RAID 1, which writes duplicate data from one drive to another; therefore, creating data redundancy and increasing fault tolerance.
RAID 5 (striping with parity)	The data in the RAID volume and parity are striped across the array's members. Parity information is written with the data in a rotating sequence across the members of the array. This RAID level is a preferred configuration for efficiency, fault-tolerance, and performance.
RAID 10 (striping and mirroring)	The RAID level where information is striped across a two drive arrays for system performance. Each of the drive in the array has a mirror for fault tolerance. RAID 10 provides the performance benefits of RAID 0 and the redundancy of RAID 1. However, it requires four hard drives so it's the least cost effective.
RAID Array	A logical grouping of physical drives.
RAID Volume	A fixed amount of space across a RAID array that appears as a single physical drive to the operating system. Each RAID volume is created with a specific RAID level to provide data redundancy or to enhance data storage performance.
Recovery Drive	The drive that is the designated target drive in a recovery volume.
RHEL	Red Hat Enterprise Linux
Intel® RSTe	Intel® Rapid Storage Technology enterprise.
RWH	It stands for RAID5 Write Hole and can cause data integrity issue
SLES	SUSE Linux Enterprise Server
TB	Terabyte
TiB	Tebibyte (1024 x 1024 x 1024 x 1024 bytes)
UEFI Mode	*Unified Extensible Firmware Interface*. Refers to the system setting in the BIOS
VIP	Intel Validation Internet Portal
Volume	This term is representative of mdadm RAID within an Intel metadata based container.
Volume initialization	Immediately after a RAID volume has been created, initialization (or resync) commences if the RAID level is 1, 10, or 5 to guarantee volume data insync
Intel® VROC	Intel® Virtual RAID on CPU

2 Enabling Intel® VMD and create/remove RAID in BIOS HII

2.1 Enabling Intel VMD in BIOS

With Intel® VROC, Intel® VMD will also need to be enabled on your new platform. This will allow for the ability to create RAID volumes within the BIOS and for bootable drives. This series of steps are provided based on an Intel® Customer Reference Board (CRB). Please refer to the instructions that have been supplied by the user's platform BIOS vendor as those instructions may differ from the set below.

Step 1: Immediately following POST, select the option that will allow the user to access the BIOS setup menu. This example uses <F2>.

Step 2: For the Intel® CRB reference BIOS, the user will want to use the arrow keys to move the cursor to the **Advanced** (it will become highlighted) and press <Enter>.

Step 3: Using the arrow keys, move the cursor to **PCI Configuration** and press <Enter>.

Step 4: Using the arrow keys, move the cursor to **Volume Management Device** and press <Enter>.

Step 5: This step varies depending on physical configuration of the system. This system has 4 direct connections from the backplane to the motherboard using Oculink cables for NVMe devices. Using the arrow keys, move the cursor to **CPU1 Oculink Volume Management** and press <Enter>. Toggle the selection from Disabled to Enable, and press <Enter> to select.

Step 6: **VMD Port 3C (PCIe SSD0)** and **VMD Port 3D (PCIe SSD1)** by default will also be Disabled. Navigate to each in turn. Press <Enter> and toggle the Disabled setting to Enabled and press <Enter> to set selection.

Step 7: Repeat steps 5 and 6 on **CPU2 Oculink Volume Management.**

Step8: Save settings by pressing <F10> to **Save Changes and Exit**.

Note: Please consult the user's Platform BIOS manufacturer documentation for a complete list of options that can be configured.

2.2 Create RAID Volume in BIOS HII

There are a number of vendors that produce equipment compatible with Intel® VROC technology, please refer to your included materials as well as the following guide materials for proper configuration. The menus may be different depending on the provider of the BIOS environment on your system. The instructions provided are based on an Intel® Customer Reference Board (CRB).

The following assumptions have been made:

1. You have enabled Intel® VMD within system BIOS.
2. You have sufficient drives of the appropriate size and type to create the RAID volume.
3. The drives are connected to the system properly per your vendor's specifications.
4. The Pre-OS is able to see all drives.

To create a RAID volume:

Step 1: To enter the Setup Menu, press the appropriate key indicated on the screen during boot up. This example uses <F2>.

Step 2: Navigate to **Advanced**, and press <Enter>

Step 3: Navigate to **PCI Configuration** and press <Enter>

Step 4: Navigate to **UEFI Option ROM Control** and press <Enter>

Step 5: Navigate to **Intel® Virtual RAID on CPU (Intel® VROC)** listed under Storage Controllers and press <Enter>

Step 6: Navigate to **All Intel® VMD Controllers** and press <Enter>

Step 7: Navigate to **Create RAID Volume** and press <Enter>

Step 8: Press <Enter> with the **Name** highlighted to modify the volume name, to keep the default, navigate to next item.

Step 9: Highlight the **RAID Level** which is by default set to RAID 0 (Stripe), press <Enter>. All available RAID selections will be presented based on the number of drives that have been installed to that set of controllers. Highlight your selection, then press <Enter> to set that value.

Step 10: To **Enable RAID spanned over VMD Controllers**, highlight the empty bracket and press <Enter>. Navigate to the X, and press the <Enter> key to set the status to "enabled". Linux does permit the use of spanned boot volumes. However, there may be a message warning that this is not supported for use for Windows.

Step 11: To select the drives to be within the array, highlight the empty bracket next to the desired drive, press <Enter>, and highlight the X. Press <Enter> to finalize selection as part of the array.

Step 12: Repeat step 11 for each drive required in the array.

Step 13: With the exception of RAID1, you may alter the **Strip Size** of your array. The default Strip Size is 64k.

Step 14: If a smaller value than 95% of total capacity is desired for the RAID, a new value may be assigned to **Capacity**. This value is in Megabytes. Highlight the current value, press <Enter>, substitute it with your preferred amount in Megabytes, and then press <Enter> to save.

Step 15: Navigate to **Create Volume** and press <Enter>

Step 16: Save changes and reboot.

2.3 Removing RAID Volumes Using the HII

In a manner much like what was used to create the volume in section 2.2 of this guide, the process to remove a RAID volume is very similar.

Step 1: As the system is booting, press the appropriate key to enter the Setup Menu. This example uses <F2>

Step 2: Navigate to **Advanced** and press <Enter>

Step 3: Navigate to **PCI Configuration** and press <Enter>

Step 4: Navigate to **UEFI Option ROM Control** and press <Enter>

Step 5: Navigate to **Intel® Virtual RAID on CPU (Intel® VROC)** listed below Storage Controllers and press <Enter>

Step 6: Navigate to the RAID to be removed below **Intel® VROC Managed Volumes** and press <Enter>

Step 7: Navigate to **Delete** and press <Enter>

Step 8: Navigate to **Yes** and press <Enter> to confirm removal of RAID.

3 Installation of Intel® VROC 6.x for Linux

Intel® VROC 6.x Linux release package for Intel® Xeon® Platforms have been included in the latest supported OSV distributions. The following instructions will provide user an opportunity to install an archive with custom components with Intel® VROC 7.0 Linux functionality that incorporates Intel VMD that is part of the Intel® Xeon® Scalable Platforms server class processors.

3.1 Supported Linux OS Distributions

VROC 7.0 Linux supported OS distributions

- RHEL 7.5 GA (VROC 6.0 PV release available in VIP)
- RHEL 7.6 GA (inbox included VROC 6.0 PV)
- RHEL 7.7 GA (inbox included VROC 6.0 PV)
- RHEL 8.0 GA (inbox included VROC 6.0 PV)
- RHEL 8.1 GA (inbox included VROC 6.0 PV)
- SLES12 SP4 GM (inbox included VROC 6.0 PV)
- SLES 12 SP5 GM (inbox included VROC 6.0 PV)
- SLES 15 GM (inbox included VROC 6.0 PV)
- SLES15 SP1 GM (inbox included VROC 6.0 PV)
- Ubuntu server 18.04.3 LTS (inbox with user guide in www.intel.com/vroc)

Note: Intel highly recommends upgrading to supported OS distribution for best VROC 6.0 experience.

3.2 Installation Files

This installation files in this section are only required for the RHEL 7.5 GA. For other supported OS distribution, they have VROC 7.0 functionality inbox in the GA/GM release. There is no additional steps or files needed for installation.

The following files are included in the Intel VROC 6.0 release kit only for RHEL 7.5 GA. The kit is named to reflect the release version and the distribution of RHEL it is intended for.

SRC directory

RPM file containing the updated ledmon source

RPM file containing the updated mdadm source

RPMS directory

The updated ledmon RPM file(s)

The updated mdadm RPM file(s)

Note: All of the supported Linux OS distribution listed in the section 3.1 except RHEL 7.5 GA include the VROC 6.0 latest ledmon and mdadm package inbox. There is no extra packages or steps needed to install VROC 6.0 on these distributions.

3.3 VROC 7.0 Installation Process

This section will focus on installing and configurating an Intel® Xeon® Scalable Platform with Intel® VMD enable and install to a SATA or NVMe device.

3.3.1 Installation VROC with Supported Linux OS Distribution Inbox

All the VROC supported Linux OS distribution (except RHEL 7.5 GA) have inbox with VROC 7.0 ledmon, mdadm, mdmon and VMD kernel drivers. There are no separate out-of-box packages needed for VROC installation. OS Boot RAID is supported after the RAID volume created in BIOS HII and install the Linux distribution with inbox installer. The detail steps to create an Intel® VROC managed RAID volume in the UEFI HII describe in Chapter 2.

3.3.2 Installation with RHEL 7.5 GA with Out-of-box VROC 6.0 PV Packages

This process will focus on installing RHEL 7.5 GA with platform that has Intel® VMD enabled NVMe passthrough drive or Intel VROC managed RAID volume. The detail steps to create an Intel® VROC managed RAID volume in the UEFI HII describe in Chapter 2.

1. Install RHEL 7.5 GA in the NVMe passthrough drive or VROC managed RAID volume
2. Copy vroc6.0_PV_rhel7.5.zip to the destination folder (available in VIP kit)
3. Go to the destination folder and upgrade ledmon/mdadm by using following command

```
# rpm –U ./ledmon-0.90-intel_5.4.el7.x86_64.rpm
# rpm –U ./mdadm-4.0-13.1.IntelVROC6.0.el7.x86_64.rpm
```

3.3.3 Loading the Intel Accelerated Storage Manager (Intel ASM)

Intel Accelerated Storage Manager (Intel ASM) is RESTful API based solution provide the system, devices, RAID information through VROC plugin and serving through built-in web server. Intel ASM packages available in VROC 6.0 PV out-of-box release package and only validated on RHEL 7.5/7.6 GA. Intel ASM is not going to support the newer Linux OS distribution beside RHEL 7.5/7.6 GA.

These instructions will outline how to install the Intel ASM tool on a new install of RHEL 7.5/7.6 GA.

1. Once RHEL 7. 5/7.6 GA has been properly booted copy the NSG_Agent-2.0.0.72-RHEL7-Release-pkg.zip to the platform
2. Unzip the file.
3. Go into the NSG_Agent-2.0.0.72-RHEL7-Release-pkg directory
4. Go to the iasm directory
5. Run “yum install ./*”
6. Go back up one directory level
7. Go to the vroc directory
8. Run “yum install ./*”
9. Go back up one directory level
10. Go to www directory
11. Run “yum install ./*”
12. Intel ASM has now been installed. Please reference the Intel ASM documentation for instructions how to configure and run the tool.

Note: This instruction set is created with the assumption that the system has been registered with RedHat and that the repos are properly loaded. Any technical requirements and reference data can be found within the Intel Accelerated Storage Manager Linux Administration Guide.

4 mdadm Quick Start Guide

Usage: #mdadm [mode] <raiddevice> [options] <component-devices>

Assemble – assemble previously created array into an active array

# mdadm –A –s	scan and assembly all the RAID volume based on config file
# mdadm –A /dev/md/md0 –e imsm /dev/<member drives>	assemble md0 container manually if without config file
# mdadm -A /dev/md/<array_name>	assemble array manually if without config file

Create/grow/Misc – Create new array with metadata/ change active size or number of active devices

# mdadm -C /dev/md/imsm0 /dev/nvme[0-3]n1 –n 4 –e imsm	Create IMSM meta data with 4 nvme0-3
# mdadm -C /dev/md/imsm0 /dev/md/md0 –n 4 –l 5	Create md0 as 4DR5 in imsm0 container
# mdadm -C /dev/md/md0 /dev/md/imsm0 – n 4 –l 5 --consistency-policy=ppl	Create md0 as 4DR5 in imsm0 container with RWH enabled
# mdadm –C /dev/md/imsm0 /dev/md/md0 –n 4 –l 5 –z $((100*1024*1024))	Create 4DR5 with 100GB size for each drive. 300GB size created with this example command
# mkfs.ext4 /dev/md/md0	Create ext4 filesystem in md0 RAID volume
# mount /dev/md/md0 /mnt/<mountpoint>	Mount md0 RAID volume in the fs
# mdadm -a /dev/md/imsm0 /dev/<nvmeXn1>	Add hot-spare drive to IMSM container
# mdadm -Ebs > /etc/mdadm.conf	export device metadata to mdadm.conf file
# mdadm –S /dev/md/md0	stop and deactivate md0 array
# mdadm –S /dev/md/imsm0	stop and deactivate imsm0 container
# mdadm –S –s	stop and deactivate all
# mdadm -f /dev/md/md0 /dev/nvme0n1	fail nvme0n1 in md0 array
# mdadm –r /dev/md/md127 /dev/nvme0n1	detached nvme0n1 in md127 container
# mdadm –-zero-superblock /dev/nvme0n1	clear and zero the superblock in nvme0n1
# export MDADM_EXPERIMENTAL=1	
# mdadm –G /dev/md/md0 –l 0	grow existing 2DR1 to 1DR0
mdadm -G /dev/md/md127 -n 2	grow 2DR0 to container
# mdadm -a /dev/md/md127 /dev/nvme2n1	add one more drive into container
# mdadm -G /dev/md/md0 -l 5 –-layout=left-asymmetric	grow 2DR0 to 3DR5
mdadm –G /dev/md/md0 –-size=128G	Grow component size md0 to the 128GB
# mdadm –G /dev/md/md0 –-consistency-policy=resync	switch RWH closure policy to resync on the running md0
# mdadm –G /dev/md/md0 –-consistency-policy=ppl	enable RWH closure mechanism on the running md0

Volume, Member, HW information

#mdadm –-detail-platform	Print detail of platform's RAID capabilities
#mdadm –D /dev/md/md0	Print detail of md0 device
#mdadm –E /dev/nvme0n1	Print contents of metadata store in nvme0n1
#cat /proc/mdstat	List all active md devices with information

Follow/Monitor/Misc – Monitor one or more devices on state changes

# systemctl status mdmonitor.service	check mdmonitor service
# systemctl start mdmonitor.service	start mdmonitor service
# systemctl stop mdmonitor.service	stop mdmonitor service
# systemctl enable mdmonitor.service	enable mdmonitor service for next boot auto start

5 Creation/Removal of RAID Volumes Using mdadm

RAID volumes can be created within the BIOS after enabling Intel® VMD by using the Intel® VROC integrated UEFI HII features, or using the mdadm command line utility, which supports the Intel® Matrix Storage Manager (IMSM) metadata format when specified with the -e imsm metadata option.

5.1 Report RAID Details from the System BIOS

Before using the mdadm to create the RAID, user could use “detail-platform” to check what Intel® VROC RAID support is provided by the BIOS:

```
# mdadm --detail-platform
       Platform : Intel(R) Virtual RAID on CPU
        Version : 6.0.0.1024
    RAID Levels : raid0 raid1 raid10 raid5
    Chunk Sizes : 4k 8k 16k 32k 64k 128k

     2TB volumes : supported

       2TB disks : supported
      Max Disks : 8
    Max Volumes : 2 per array, 8 per controller
 I/O Controller : /sys/devices/pci0000:00/0000:00:11.5 (SATA)
          Port0 : - no device attached -
          Port1 : - no device attached -
          Port2 : - no device attached -
          Port3 : - no device attached -

          Port4 : - no device attached -

          Port5 : - no device attached -


       Platform : Intel(R) Virtual RAID on CPU

        Version : 6.0.0.1024

    RAID Levels : raid0 raid1 raid10 raid5

    Chunk Sizes : 4k 8k 16k 32k 64k 128k

    2TB volumes : supported

      2TB disks : supported

      Max Disks : 8

    Max Volumes : 2 per array, 8 per controller

 I/O Controller : /sys/devices/pci0000:00/0000:17.0 (SATA)

          Port4 : /dev/sda (W629KMP2)

          Port0 : - no device attached -

          Port1 : - no device attached -

          Port2 : - no device attached -

          Port3 : - no device attached -

          Port5 : - no device attached -

          Port6 : - no device attached -

          Port7 : - no device attached -
```

```
      Platform : Intel(R) Virtual RAID on CPU
       Version : 6.0.0.1024
   RAID Levels : raid0 raid1 raid10 raid5
   Chunk Sizes : 4k 8k 16k 32k 64k 128k
   2TB volumes : supported
     2TB disks : supported
     Max Disks : 24
   Max Volumes : 2 per array, 24 per controller
 3rd Party NVMe : supported
 I/O Controller : /sys/devices/pci0000:85/0000:85:05.5 (VMD)
 I/O Controller : /sys/devices/pci0000:17/0000:17:05.5 (VMD)
 I/O Controller : /sys/devices/pci0000:5d:0000:5d:05.5 (VMD)
 NVMe under VMD : /sys/devices/pci0000:5d/0000:5d.05.5/pci10001:00/10001:00:02.0/10001:01:00.0
 NVMe under VMD : /sys/devices/pci0000:5d/0000:5d:05.5
/pci10001:5d.05.5/pci10001:00:03.0/10001:02:00.0
 I/O Controller : /sys/devices/pci0000:d7/0000:d7:05.5 (VMD)
 NVMe under VMD : /sys/devices/pci0000:d7/0000:d7:05.5/pci10004:00/10004:00:00.0/10004:01:00.0
 NVMe under VMD : /sys/devices/pci0000:d7/0000:d7:05.5/pci10004:00:01.0/10004:02:00.0
 I/O Controller : sys/devices/pci0000:ae:05.5 (VMD)
```

Note: The version represented in this output is the version of the platforms Intel VROC PreOS version, not to be confused with the version of mdadm used.

5.2 RAID Volume Creation

WARNING: *Creating a RAID array will permanently delete any existing data on the selected drives. Back up all important data before beginning these steps.*

The following shows an example of how to create a RAID5 array with 4 Intel NVMe SSDs:

1. First, a container that establishes Intel® IMSM metadata **must** be created.

```
# mdadm -C /dev/md/imsm0 /dev/nvme[0-3]n1 -n 4 -e imsm
Continue creating array? Y
mdadm: container /dev/md/imsm0 prepared.
```

The command creates a RAID container with Intel® Matrix Storage Manager metadata format. The device node for the container will be `/dev/md/imsm0`. In this example drives nvme0n1, nvme1n1 nvme2n1 and nvme3n1 are used for this RAID container and the total number of drives is 4. The wildcard expression `/dev/nvme[0-3]n1` can be used to specify the range of drives. Individual drives can be also used. The wildcard expression was parsed by the shell. If positional parameter consisting of more than a single digit, it must be enclosed in braces. Eg: /dev/nvme{9,10}n1

2. Next, a RAID 5 volume is created.

```
# mdadm -C /dev/md/md0 /dev/md/imsm0 -n 4 -l 5
```

The command creates a RAID 5 volume /dev/md0 within the `/dev/md/imsm0` container.

The following command parameters may also be used to give finer control for the creation of the RAID volume.

`-n` Number of active RAID devices to be used in the volume.

`-c` Specifies the chunk (strip) size in Kilobytes. The default is 128KiB. This value must be a multiple of 4KiB. Optimal chunk size should be considered depending on expected workload profiles.

`-l` specifies the RAID level. The options supported are 0, 1, 5, 10.

`-z` Specifies the size (in Kibibyte) of space dedicated on each disk to the RAID volume. This must be a multiple of the chunk size. For example:

```
# mdadm -C /dev/md/md0 /dev/md/imsm0 -n 4 -l 5 -z $((100*1024*1024))
```

The command above creates a RAID volume inside the /dev/md/imsm0 container with 100GiB of disk space used on each drive member.

A suffix of 'M' or 'G' can be given to indicate Mibi bytes or Gibi bytes respectively. This applies also to the –c parameter. So the above command is equivalent to this:

```
# mdadm -C /dev/md/md0 /dev/md/imsm0 -n 4 -l 5 -z 100G
```

5.3 Additional RAID Container and Volume Creation Examples

Create a 2 drive container and a RAID 0 volume

```
# mdadm -C /dev/md/imsm0 /dev/nvme[0-1]n1 -n 2 -e imsm
# mdadm -C /dev/md/md0 /dev/md/imsm0 -n 2 -l 0
```

Create a 2 drive container and a RAID 1 volume

```
# mdadm -C /dev/md/imsm0 /dev/nvme[0-1]n1 -n 2 -e imsm
# mdadm -C /dev/md/md0 /dev/md/imsm0 -n 2 -l 1
```

Create a 3 drive container and a RAID 5 volume

```
# mdadm -C /dev/md/imsm0 /dev/nvme[0-2]n1 -n 3 -e imsm
# mdadm -C /dev/md/md0 /dev/md/imsm0 -n 3 -l 5
```

Create a 4 drive container and a RAID 10 volume

```
# mdadm -C /dev/md/imsm0 /dev/nvme[0-3]n1 -n 4 -e imsm
# mdadm -C /dev/md/md0 /dev/md/imsm0 -n 4 -l 10
```

Note: To create multiple RAID volumes in the same container, they MUST span equal number of drives. For example, in order to have a RAID 0 volume and a RAID 5 volume in the same container, four drives are used for both volumes.

5.4 Adding a Spare

Adding a spare drive allows for immediate reconstruction of the RAID volume when a device failure is detected. mdadm will mark the failed device as "bad" and start reconstruction with the first available spare drive. The spare drive can also be used to grow the RAID volume (refer to section 6 for Online Capacity Expansion information). The spare drives sit idle during normal operations. When using mdadm with imsm metadata, the spare drive added to a container is dedicated to that specific container. The following command adds a spare drive to the designated container */dev/md*/imsm0

```
# mdadm -a /dev/md/imsm0 /dev/<nvmeXn1>
```

Note: The metadata UUID for the member drive inside the container as spare will keep to be all zero until the drive re-assign be part of the RAID volume. Member drive with zero UUID may be allocated to separate container after system reboot.

5.5 Creating RAID Configuration File

A configuration file can be created to record the existing RAID volumes. The information can be extracted from the existing RAID setup. The configuration file is typically stored at the default location of /etc/mdadm.conf. This allows a consistent assembling links in /dev/md after platform boots. This reliable and constant link can be used in system (e.g. in /etc/fstab for automounting). The assembling nodes in the /dev may change depend on system enumeration.

```
# mdadm -Ebs > /etc/mdadm.conf
```

Depending on OS distribution, there may be alternative means of saving the configuration file. It is important to reference the mdadm.conf man page to determine what applies to your distribution.

5.6 RAID Volume Initialization/Resync

Immediately after a RAID volume has been created, initialization (or resync) commences if the RAID level is 1, 10, or 5. During this time, any data stored on RAID level 5 volumes are not guaranteed to be safe if failure occurs. If a drive failure happens during the initialization time, recovery will not be possible. This scenario is also true during RAID volume rebuilds.

5.6.1 Adjusting Initialization/Resync Speed

Linux by default has a speed of 200MB as the speed for volume assembly and synchronization. This was the community standard set based on spinning platter based hard drives. A consensus has not been reached regarding what the standard should be moved to with the advancements in technology.

This value can be modified to prevent the system from taking an undue amount of time initializing or rebuilding a RAID array. The series of commands below shows a system that is set at the default value, and how that may be modified.

To see what the current rate of speed the system is using:

```
# cat /sys/block/md0/md/sync_speed_max

200000(system)
```

To modify this to a higher value, such as 5GB for NVMe drives:

```
# echo 5000000 > /sys/block/md0/md/sync_speed_max
```

Note: Different values may be appropriate for different drive types. 5GB is reasonable for Data Center Intel NVMe SSD models.

5.7 Removing RAID Volumes Using mdadm

RAID arrays can be built using mdadm, but they can also be removed via mdadm.

First, stop the array. To stop all active RAID volumes, the following command can be used. mdadm will scan for and stop all running RAID volumes and containers.

```
# mdadm –vS /dev/md/md0

mdadm: stopped /dev/md/md0

# mdadm –S –s
```

And to stop a container, the following command can be used.

```
# mdadm -S /dev/md/imsm0
```

Now check the status of the array:

```
# mdadm -D /dev/md/md0

mdadm cannot open /dev/md/md0: No such file or directory
```

Then, remove or delete the superblocks from all associated disks with the following command:

```
# mdadm --zero-superblock /dev/nvmeXn1 /dev/nvmeYn1 /dev/nvmeZn1
```

5.8 Removing an Array Within a Matrix

If there are more than one arrays within a container, this is referred to as a Matrix RAID. This requires a little more care in the implementation of removal.

First, both matrix volume must be stopped:

```
# mdadm -S /dev/md/md{numberForFirstArray}

# mdadm -S /dev/md/md{numberForSecondArray}
```

Next, remove the volume metadata:

```
# mdadm --kill-subarray={index} /dev/md/md{container_number}
```

Where {index} – should be either 0 or 1 to represent the first or second volume. The following command will show the subarray index in the container.

Note: The UUID and index number will be changed after subarray deletion. If user delete index 1 subarray, index 0 will keep in the container. If user delete index 0 subarray, index 1 will become index 0 after deletion.

```
# mdadm -D /dev/md/md{RAIDArray}
```

The other way to get the subarray index via the mdstat in following example show the index after volume

```
Personalities : [raid0]

md125 : active raid0 sdc[1] sdb[0]

      58699776 blocks super external:/md127/1 128k chunks

md126 : active raid0 sdc[1] sdb[0]

      4194304 blocks super external:/md127/0 128k chunks

md127 : inactive sdc[1](S) sdb[0](S)

      2210 blocks super external:imsm
```

5.9 Erasing RAID Metadata

Having incorrect or bad metadata can cause RAID volumes to be assembled incorrectly. The metadata can be erased on each potential member drive with the following command to ensure the drive is clean. This operation does not attempt to wipe existing user data but, will delete an array if it's a current member drive or spare drive. The RAID volume have to be stopped and deactivated to run the erase operation.

```
# mdadm --zero-superblock /dev/<nvmeXn1>
```

Multiple drives can be specified to clear the superblock at the same time.

Note: The metadata erase operation is irreversible and may ruin the RAID volume. Please run the operation with caution.

6 RAID Reporting and Monitoring

6.1 Reporting RAID Information

Use the following command, to print out details about a RAID container or volume:

```
# mdadm -D /dev/md/imsm0
/dev/md/imsm0:
        Version : imsm
     Raid Level : container
  Total Devices : 4

Working Devices : 4
          UUID : b559b502:b199f86f:ee9fbd40:cd10e91d
  Member Arrays :

    Number   Major   Minor   Raid Device
       0       8       32        -        /dev/nvme0n1
       1       8       48        -        /dev/nvme1n1
       2       8       80        -        /dev/nvme2n1
       3       8       96        -        /dev/nvme3n1
```

The command –D will print all that the kernel knows about the array in its current state.

```
# mdadm -D /dev/md/md0
/dev/md0:
      Container : /dev/md/imsm0, member 0
     Raid Level : raid1
     Array Size : 781408256 (745.21 GiB 800.16 GB)
  Used Dev Size : 781409024 (745.21 GiB 800.16 GB)
   Raid Devices : 2
  Total Devices : 2


          State : clean, resyncing
 Active Devices : 2
Working Devices : 2
 Failed Devices : 0
  Spare Devices : 0


  Resync Status : 38% complete

           UUID : 084d2b20:09897744:36757c5b:77e0e945
    Number   Major   Minor   Raid Device State
       0     259       2        0      active sync   /dev/nvme1n1
       1     259       1        1      active sync   /dev/nvme0n1
```

To print out metadata state details about a member drive:

```
# mdadm -E /dev/nvme0n1
/dev/nvme1n1:
          Magic : Intel Raid ISM Cfg Sig.
        Version : 1.1.00
    Orig Family : f957b05e
         Family : f957b05e
     Generation : 00000015
     Attributes : All supported
           UUID : 5f85c04a:be746f31:67d16085:cff460f1
       Checksum : abb8c53b correct
    MPB Sectors : 1
          Disks : 2
   RAID Devices : 1
  Disk00 Serial : LJ7423090T1P0FGN
          State : active
             Id : 00000000
    Usable Size : 1953514766 (931.51 GiB 1000.20 GB)
[1]:
           UUID : f196d4f7:81aea65f:f7391cae:ddfd67a7
     RAID Level : 1
        Members : 2
          Slots : [UU]
    Failed disk : none
      This Slot : 0
    Sector Size : 512
     Array Size : 33554432 (16.00 GiB 17.18 GB)
   Per Dev Size : 33556480 (16.00 GiB 17.18 GB)
  Sector Offset : 0
    Num Stripes : 131072
     Chunk Size : 64 KiB
       Reserved : 0
  Migrate State : idle
      Map State : normal
    Dirty State : clean
     RWH Policy : off
  Disk01 Serial : FT5462000H1P6JGN
          State : active
             Id : 00000000
    Usable Size : 3125617166 (1490.41 GiB 1600.32 GB)
```

To get the most current status on all RAID volumes, the file `/proc/mdstat` can be examined. This file is a special file that is updated continuously to show the status of all the containers, and RAID volumes. In the example below, the status shows that currently available supported RAID levels are 0, 1, 5 and 10. Md0 is the active RAID volume with RAID level 5 and 128k strip size. The RAID volume contains 4 drives that are all in normal (UP) status. “md127” is the IMSM container for the RAID volume.

```
# cat /proc/mdstat

Personalities : [raid1] [raid6] [raid5] [raid4]

md0 : active raid5 nvme0n1[3] nvme3n1[2] nvme2n1[1] nvme1n1[0]

      2930270208 blocks super external:/md127/0 level 5, 128k chunk, algorithm 0 [4/4] [UUUU]

md127 : inactive nvme3n1[3](S) nvme2n1[2](S) nvme1n1[1](S) nvme0n1[0](S)

      4420 blocks super external:imsm

unused devices: <none>
```

Note: When creating containers and volumes, you will notice that in /proc/mdstat the device names will not match up exactly. For example, when /dev/md/imsm0 is created, md127 will be shown in /proc/mdstat and other detail output as well. The /dev/md/imsm0 is created as an alias of /dev/md127 device node. Looking in the /dev/md directory, one will notice that /dev/md/imsm0 is a soft link to /dev/md/md127.

6.2 Logging

Various messages coming from MDRAID subsystem in the kernel are logged. Typically the messages are stored in the log file `/var/log/messages` in popular Linux distributions with other kernel status, warning, and error outputs.

Below is an example snippet of what the log may look like:

```
Dec 11 15:07:28 localhost kernel: md/raid:md0: not clean -- starting background reconstruction

Dec 11 15:07:28 localhost kernel: md/raid:md0: device nvme0n1 operational as raid disk 3

Dec 11 15:07:28 localhost kernel: md/raid:md0: device nvme3n1 operational as raid disk 2

Dec 11 15:07:28 localhost kernel: md/raid:md0: device nvme2n1 operational as raid disk 1

Dec 11 15:07:28 localhost kernel: md/raid:md0: device nvme1n1 operational as raid disk 0

Dec 11 15:07:28 localhost kernel: md/raid:md0: raid level 5 active with 4 out of 4 devices,
algorithm 0

Dec 11 15:07:28 localhost kernel: md/raid456: discard support disabled due to uncertainty.

Dec 11 15:07:28 localhost kernel: Set raid456.devices_handle_discard_safely=Y to override.

Dec 11 15:07:28 localhost kernel: md0: detected capacity change from 0 to 3000596692992

Dec 11 15:07:28 localhost systemd: Starting MD Metadata Monitor on /dev/md127...

Dec 11 15:07:28 localhost systemd: Started MD Metadata Monitor on /dev/md127.

Dec 11 15:07:28 localhost kernel: md: resync of RAID array md0
```

There are two components within the mdadm tools to monitor events for the RAID volumes. Mdadm (with –monitor” parameter) can be used to monitor general RAID events, and mdmon provides the ability to monitor “metadata event” occurrences such as drive failures, clean-to-dirty transitions, etc. for external metadata based RAID volumes. The kernel provides the ability to report such actions to the user space via sysfs, and mdmon takes action accordingly with the monitoring capability. The mdmon polls the sysfs looking for changes in the entry value for, *array_state*, *sync_action*, and per drive *state* attribute files.

6.3 mdmon

The mdadm monitor, mdmon, is automatically started when MDRAID volumes are activated by mdadm through creation or assemble. Mdmon is metadata manager and updater. The daemon can be started manually as shown below:

1. Run the following command:

```
# mdmon /dev/md0
```

The --all parameter can be used in place of the container name to star monitors for all active containers.

Note: mdmon must be started in the initramfs in order to support an external metadata RAID array as the root file system. mdmon needs to be restarted in the new namespace once the final root files system has been mounted. It's not recommend to manual start or re-start the daemon.

```
# mdmon --takeover --all
```

6.4 Monitoring Using mdadm and systemctl

All monitoring services designed with a fresh install of RHEL 7.X to initialize upon creation of the RAID volume, however, there are methods to ensure that these functions have initialized properly. These tools will allow for the Linux administrator to check and enable tools on the system for use to assist in RAID monitoring within the CLI environment.

6.4.1 mdadm Monitoring Daemon

mdadm monitoring can be started with the following command line:

```
# mdadm --monitor --scan --daemonise --syslog
```

The command above runs mdadm as a daemon to monitor all md devices. All events will be reported to syslog. The user can monitor the syslog and filter for specific mdadm events generated. It is mandatory to define the email address in the mdadm.conf file before start the monitoring.

6.4.2 systemctl for RAID Monitoring

To see if the mdmonitor services are running as anticipated, you may use the following command to check the status:

```
# systemctl status mdmonitor.service
```

The command below can be used below to manually start the service if you find that it is not running:

```
# systemctl start mdmonitor.service
```

Alternatively, the service can be enabled to start immediately following each reboot. Use this command:

```
# systemctl enable mdmonitor.service
```

To stop the service, this command is used.

```
# systemctl stop mdmonitor.service
```

The following table lists additional command line parameters that can be passed to mdadm at startup.

Table 6: Parameters for mdadm in Monitor Mode

Long form	Short form	Description
`--mail`	`-m`	Provide mail address to email alerts or failures to.
`--program or --alert`	`-p`	Provide program to run when an event is detected.
`--delay`	`-d`	Seconds of delay between polling state. Default is 60s.
`--config`	`-c`	Specify a different config file.
`--scan`	`-s`	Find mail-address/program settings in config file.
`--oneshot`	`-1`	Check for degraded arrays and then exit.
`--test`	`-t`	Generate a Test Message event against each array at startup.
`--syslog`	`-y`	Cause all events to be reported through 'syslog'. The messages have facility of 'daemon' and varying priorities.
`--increment`	`-r`	Give a percentage increment. mdadm will generate RebuildNN events with the NN indicating the percentage at which the rebuild event had happened.
`--daemonise`	`-f`	Run as background daemon if monitoring.
`--pid-file`	`-i`	Write the pid of the daemon process to specified file.
`--no-sharing`	`N/A`	This inhibits the functionality for moving spare drives between arrays.

Note: mdmonitor service have to re-start if mdmonitor.service or mdadm.conf file change.

The following table presents all the events that are reported by mdadm monitor:

Table 7: Monitoring Events

Event Name	Description
DeviceDisappeared	An MD array previously configured no longer exists.
RebuildStarted	An MD array started reconstruction.
RebuildNN	NN is a 2 digit number that indicates rebuild has passed that many percent of the total. For example, Rebuild50 will trigger an event when 50% of rebuild has completed.
RebuildFinished	An MD array has completed rebuild.
Fail[1]	An active component of an array has been marked faulty.
FailSpare[1]	A spare drive that was being rebuilt to replace a faulty device has failed.
SpareActive	A spare drive that was being rebuilt to replace a fault device is rebuilt and active.
NewArray	A new MD array has been detected in /proc/mdstat.
DegradedArray[1]	A newly discovered array appears to be degraded.
MoveSpare	A spare drive has been moved from one array in a spare group to another array to replace a failed drive. Both arrays are labeled with the same spare group.
SparesMissing[1]	The spare device(s) does not exist in comparison to the config file when the MD array is first discovered.
TestMessage[1]	Discovered new array while --test flag was used.

Note: 1. The events indicated cause email to be sent. These can also be sent to the syslog, examples can be located in section 6.6.

6.5 Configuration File for Monitoring

mdadm will check the mdadm.conf configuration file to extract the appropriate entries for monitoring. The following entries we can set to pass to mdmon:

- MAILADDR: This configuration entry allows an E-mail address to be used for alerts. Only one email address should be used.
- MAILFROM: This configuration entry sets the email address to appear from the alert emails. The default from would be the “root” user with no domain. This entry overrides the default.
- PROGRAM: This configuration entry sets the program to run when mdmon detects potentially interesting events on any of the arrays it is monitoring. There can be only one PROGRAM line in the configuration file.

6.6 Examples of Monitored Events in syslog

```
Personalities : [raid5]

md0 : active raid5 nvme2n1[2] nvme1n1[1] nvme0n1[0]
      204800 blocks super external:/md127/0 level 5, 128k chunk, algorithm 0 [3/3] [UUU]

md127 : inactive - nvme2n1[2](S) nvme1n1[1](S) nvme0n1[0](S)
      3315 blocks super external:imsm
unused devices: <none>
```

In order to monitor all RAID containers a mdadm daemon can be started using the following command:

```
# mdadm --monitor --scan --daemonise -syslog
```

Note: only one monitor daemon can be run at the same time. User have to kill existing daemon before run the new one.

Note: make sure that for RHEL based systems user need to specify the MAILADDR in mdadm config file in order to run mdmonitor service (simply: #echo “MAILADDR root” >> /etc/mdadm.conf)

All events now will be written to syslog. After a mdadm daemon has been started the following messages can be found in /var/log/messages or the corresponding syslog file the distribution has designated:

```
May 15 09:58:40 myhost mdadm[9863]: NewArray event detected on md device /dev/md0
May 15 09:58:40 myhost mdadm[9863]: NewArray event detected on md device /dev/md127
```

When a spare drive has been added:

```
May 15 09:59:07 myhost mdadm[9863]: SpareActive event detected on md device /dev/md0, component
device /dev/<nvmeXn1>
```

When an OLCE command is finished:

```
May 15 09:59:16 myhost mdadm[9863]: RebuildFinished event detected on md device /dev/md0
```

When a drive fails:

```
May 15 10:01:04 myhost mdadm[9863]: Fail event detected on md device /dev/md0, component device
/dev/<nvmeXn1>
```

When a rebuild finishes:

```
May 15 10:02:22 myhost mdadm[9863]: RebuildFinished event detected on md device /dev/md0
May 15 10:02:22 myhost mdadm[9863]: SpareActive event detected on md device /dev/md0
```

When all MD devices are stopped:

```
May 15 10:03:27 myhost mdadm[9863]: DeviceDisappeared event detected on md device /dev/md0
May 15 10:03:27 myhost mdadm[9863]: DeviceDisappeared event detected on md device /dev/md127
```

7 Additional Volume Operations with mdadm

7.1 Volume Assembly

RAID volumes can be assembled on a supported system with a supported operating systems using the mdadm application. All inactive RAID volumes can be activated using the assemble option with mdadm.

The following command scans for the mdadm configuration file at `/etc/mdadm.conf` in order to assemble the RAID volumes. If the configuration file is not found, it scans all available drives for RAID members and assembles all the RAID volumes. And /dev/md link add "_0" to each volume as suffix.

```
# mdadm -A -s
```

To manually assemble and activate RAID volumes without the configuration file, the following example can be used:

```
# mdadm -A /dev/md/md0 -e imsm /dev/<member drives>
```

The command assembles the container with the name `/dev/md/md0` using the provided list of drives.

For additional information on mdadm.conf, consult the Linux man pages.

7.2 File System Creation on a RAID Volume

After the RAID volume has been created, a file system can be created in order to allow the mounting of the RAID volume.

```
# mkfs.ext4 /dev/md/md0
```

After the file system has been created, it can be mounted to the location of choice:

```
# mount /dev/md/md0 /mnt/<mountpoint>
```

7.3 Online Capacity Expansion

The Online Capacity Expansion (OLCE) feature allows the capacity expansion of the RAID volumes. With the "online" feature, the operation can be performed while a file system is mounted on top of the RAID volume. This avoids having down time from taking the RAID volume offline for service or loss of data.

The size of a RAID volume can be increased by adding additional drives to the RAID container or (only if it is the last volume in the container) by expanding it on existing unused drive space available to the RAID volume. In the first case if two volumes exist in the same container, OLCE is performed automatically on both volumes (one by one) because of the requirement that all volumes must span the same set of disks for IMSM metadata.

The following commands can be issued to grow the RAID volume. The first assumes that it is the last volume in the container and we have additional room to grow, and the second assumes that an additional drive has been added to the IMSM container.

Warning: *Even though the data remains intact, data should always be backed up before performing expansion operations.*

If there is additional room in the last volume of the container, the volume can be grown to the maximum available capacity.

Note: This feature is only available starting with mdadm v3.2.5. It is guarded by the MDADM_EXPERIMENTAL environment variable which must be set to "1" for GROW command to success.

```
# mdadm -G /dev/md/md0 --size=max
```

Note: RAID0 size expansion is not current supported

The example below adds a single drive to the RAID container and then grows the volume(s). Because IMSM volumes inside a container must span the same number of drives, all volumes are expanded. A backup file that MDRAID will store the backup superblock is specified. This file must not reside on any of the active RAID volumes that are being worked on.

```
# mdadm -a /dev/md/imsm0 /dev/<nvmeXn1>
# mdadm -G /dev/md/md0 -n 4 --backup-file=/tmp/backup
```

Note: The operation may depend on the different RAID level for variant member drives. Refer to Chapter 6.4 for more detail.

7.4 Fail and Remove Drive

To mark an active drive as, “failed” (or set as faulty) manually, basically two steps are used to remove failed drive from the volume and container.

First step is to fail the drive from the volume. The failed drive still stays in the container but not in the RAID volume.

```
# mdadm -f /dev/md/md0 /dev/<nvmeXn1>
```

The second step is to remove a failed drive from the container, the following command needs to be executed. This only works on a container based RAID volume.

```
# mdadm -r /dev/md/md127 /dev/<nvmeXn1>
```

Note: Once the logical drive is removed from the volume with the command above the system needs to be powered off before removing the physical NVMe SSD from the system.

7.5 Raid Level Migration

The RAID level migration feature allows changing of the RAID volume level without loss of data stored on the volume. It does not require re-installation of the operating system. All applications and data remain intact.

Warning: *Even though the data remains intact, data should always be backed up before performing migration operations.*

The following table shows the available migration support with Intel® IMSM metadata. You must have the appropriate number of drives necessary for the level you’re converting to as spare drives.

Table 8: Migration Capabilities with IMSM

Destination → / ↓ Source level	RAID 0	RAID 1	RAID 10	RAID 5
RAID 0	N/A	No	**Yes**	**Yes**
RAID 1	**Yes**	N/A	No	***Yes**
RAID 10	**Yes**	No	N/A	***Yes**
RAID 5	No	No	No	N/A

Note: Migrations from RAID 1 to RAID 5 or from RAID 10 to RAID 5 must be done in two steps. A conversion to RAID 0 first is necessary before converting to RAID 5. During the second step (migration from RAID 0 to RAID 5) the addition of spare drive(s) may be needed.

Before performing any grow operations (if not already done), one environmental variable needs to be set:

```
# export MDADM_EXPERIMENTAL=1
```

Note: This feature is only available starting from mdadm v3.2.5. This setting is case sensitive

The following is an example of migration from 2 disk RAID 1 to 3 disk RAID 5:

```
# cat /proc/mdstat

Personalities : [raid6] [raid5] [raid4] [raid1]
md0 : active raid1 nvme0n1[1] nvme1n1[0]
      16777216 blocks super external:/md127/0 [2/2] [UU]

md127 : inactive nvme1n1[1](S) nvme0n1[0](S)
      2210 blocks super external:imsm

unused devices: <none>
```

First step is to migrate from 2 disk RAID 1 to 1 disk RAID 0:

```
# mdadm –G /dev/md0 –l 0

md/raid1:md0: Disk failure on nvme1n1, disabling device.

md/raid1:md0: Operation continuing on 1devices.

mdadm: level of /dev/md0 changed to raid0

Personalities : [raid1] [raid0] [raid6] [raid5] [raid4]
md0 : active raid0 nvme0n1[1]
      16777216 blocks super external:/md127/0 64k chunks

md127 : inactive nvme1n1[1](S) nvme0n1[0](S)
      2210 blocks super external:imsm

unused devices: <none>
```

Second step : Use Online Capacity Expansion to go from 1 disk RAID 0 to 2 disk RAID 0:

```
# mdadm -G /dev/md127 -n 2
# cat /proc/mdstat
Personalities : [raid1] [raid0] [raid6] [raid5] [raid4]
md0 : active raid4 nvme1n1[2] nvme0n1[1]
      16777216 blocks super external:/md127/0 level 4, 64k chunks algorithm 0 [3/2] [U__]

md127 : inactive nvme1n[1](S) nvme0n1[0](S)
      2210 blocks super external:imsm

unused devices: <none>
```

Note: This is an intermediate step, while Linux supports RAID 4, it is not supported as a final stage solution by Intel VROC. Please proceed to the next step to ensure preservation of your data.

Third step : Add a spare disk to the container md127:

```
# mdadm -a /dev/md127 /dev/nvme2n1
# cat /proc/mdstat
Personalities : [raid0] [raid1] [raid6] [raid5] [raid4]
md0 : active raid0 nvme1n1[2] nvme0n1[1]
      16777216 blocks super external:/md127/0 64k chunks

md127 : inactive – nvme2n1[2](S) nvme1n1[1](S) nvme0n1[0](S)
      3315 blocks super external:imsm

unused devices: <none>
```

Forth step: Migrate from 2 disk RAID 0 to 3 disk RAID 5:

```
# mdadm -G /dev/md0 -l 5 --layout=left-asymmetric
# cat /proc/mdstat
Personalities : [raid0] [raid1] [raid6] [raid5] [raid4]
md0 : active raid5 nvme2n1[3] nvme1n1[2] nvme0n1[1]
      16777216 blocks super external:/md127/0 level 5, 64k chunk, algorithm 0 [3/3] [UUU]

md127 : inactive - nvme2n1[2](S) nvme1n1[1](S) nvme0n1[0](S)
      3315 blocks super external:imsm

unused devices: <none>
```

Warning: *IMSM metadata only supports the left-asymmetric layout of RAID 5. The default layout is left-symmetric, so during migrations the layout for IMSM metadata has to be specified explicitly. Left-asymmetric is the use of a rotating parity 0 with data continuation.*

Note: The command to adjust consistency policy will fail during the reshape process is in process. The message "mdadm: PPL cannot be enabled when reshape is in progress" will be presented.

This is another example to migrate from 4 disk RAID 10 to 4 disk RAID5

```
[root@localhost ~]# cat /proc/mdstat
Personalities : [raid1] [raid10]
md1 : active raid10 nvme2n1[3] nvme3n1[2] nvme0n1[1] nvme1n1[0]
      20971520 blocks super external:/md127/0 128K chunks 2 near-copies [4/4] [UUUU]
md127 : inactive nvme3n1[3](S) nvme2n1[2](S) nvme1n1[1](S) nvme0n1[0](S)
      4420 blocks super external:imsm
unused devices: <none>
```

First step is to migrate for 4 disk RAID 10 to 2 disk RAID 0

```
[root@localhost ~]# mdadm -G /dev/md1 -l 0
mdadm: level of /dev/md1 changed to raid0
[root@localhost ~]# cat /proc/mdstat
Personalities : [raid1] [raid10] [raid0]
md1 : active raid0 nvme2n1[3] nvme1n1[0]
      20971520 blocks super external:/md127/0 128k chunks
md127 : inactive nvme3n1[3](S) nvme2n1[2](S) nvme1n1[1](S) nvme0n1[0](S)
      4420 blocks super external:imsm
unused devices: <none>
```

Second step is to migrate from 2 disk RAID0 to 3 disk RAID 5

```
[root@localhost ~]# mdadm -G /dev/md1 -l 5 --layout=left-asymmetric
mdadm: level of /dev/md1 changed to raid5
[root@localhost ~]# cat /proc/mdstat
Personalities : [raid1] [raid10] [raid0] [raid6] [raid5] [raid4]
md1 : active raid5 nvme3n1[4] nvme2n1[3] nvme1n1[0]
      20971520 blocks super external:-md127/0 level 5, 128k chunk, algorithm 5 [3/2] [UU_]
      [==>..................]  reshape = 13.9% (1458176/10485760) finish=13.9min speed=10768K/sec
md127 : inactive nvme3n1[3](S) nvme2n1[2](S) nvme1n1[1](S) nvme0n1[0](S)
      4420 blocks super external:imsm
unused devices: <none>
```

Third step is to expand from 3 disk RAID5 to 4 disk RAID 5. Resync have to be completed before reshape

```
[root@localhost ~]# mdadm -G /dev/md127 -n 4
[root@localhost ~]# cat /proc/mdstat
Personalities : [raid1] [raid10] [raid0] [raid6] [raid5] [raid4]
md1 : active raid5 nvme0n1[5] nvme3n1[4] nvme2n1[3] nvme1n1[0]
      20971520 blocks super external:-md127/0 level 5, 128k chunk, algorithm 0 [4/4] [UUUU]
      [>.....................]  reshape =  3.9% (416256/10485760) finish=1.2min speed=138752K/sec
md127 : inactive nvme3n1[3](S) nvme2n1[2](S) nvme1n1[1](S) nvme0n1[0](S)
      4420 blocks super external:imsm
unused devices: <none>
```

7.6 Freezing Reshape

If a RAID volume is in the process of reshape, the reshape process will be frozen during the initramfs booting phase and will resume when the system is up and running. Starting with mdadm 3.2.5 these features are supported. Distributions from the Operating System Vendors should have taken cared of this in their init script setup utilities, but details are described below for customers that are building their own distribution.

Warning: *Even though the data remains intact, data should always be backed up before performing reshape operations.*

The parameters `--freeze-reshape` is used to pause the reshape operation during system start up initramfs phase. For example:

```
# mdadm -As --freeze-reshape
```

When reshape is frozen, the status provided by /proc/mdstat will denote the state with a hyphen such as “super external:-md127/0” instead of “super external:/md127/0”:

```
Personalities : [raid5]
md0 : active raid5 nvme2n1[2] nvme1n1[1] nvme0n1[0]
      204800 blocks super external:-md127/0 level 5, 128k chunk, algorithm 0 [3/3] [UUU]
      [>.....................]  reshape =  2.2% (116736/5242880) finish=501934.9min speed=0K/sec

md127 : inactive -
      9459 blocks super external:imsm

unused devices: <none>
```

Once the system is up, the following example with the parameter `--continue` can be used to resume the reshape process:

```
# mdadm -G /dev/md/md0 --continue
```

Or with a volume:

```
# mdadm -G /dev/md/Volume0 --continue
```

7.7 RAID Write Hole Example Protection

RAID write hole (RWH) stands for RAID5 write hole. A system crash or other interruption of a write operation can result in states where the parity is inconsistent with the data due to non-atomicity of the write process, such that the parity cannot be used for recovery in the case of a disk failure (the so-called RAID 5 write hole). The RAID write hole is a known data corruption issue in older and low-end RAIDs, caused by interrupted de-staging of writes to disk.

Partial Parity Log (PPL) is a feature available for RAID5 arrays. Partial parity for a write operation is the XOR of stripe data chunks not modified by this write. It is just enough data needed for recovering from the write hole. XORing

partial parity with the modified chunks produces parity for the stripe, consistent with its state before the write operation, regardless of which chunk writes have completed. If one of the not modified data disks of this stripe is missing, this updated parity can be used to recover its contents. PPL recovery is also performed when starting an array after an unclean shutdown and all disks are available, eliminating the need to resync the array.

VROC support RWH closure with PPL. The consistency policy of an active array can be changed by using the "consistency-policy" option in the create or grow mode. Current mdadm support for ppl and resync policies to enable or disable the RAID5 partial parity log.

To create a RAID5 volume with enabled RWH policy:

```
# mdadm -C /dev/md/imsm0 /dev/nvme[0-3]n1 -n 4 -e imsm

Continue creating array? Y
mdadm: container /dev/md/imsm0 prepared.

# mdadm -C /dev/md0 /dev/md/imsm0 -l5 -n4 --consistency-policy=ppl
```

To check the current RWH policy:

```
# mdadm -D /dev/md0

/dev/md0:

     Container : /dev/md/imsm0, member 0

    Raid Level : raid5

    Array Size : 117212672 (1117.83 GiB 1200.26 GB)

 Used Dev Size : 390708224 (372.61 GiB 400.09 GB)

  Raid Devices : 4

 Total Devices : 4


         State : clean, resyncing

 Active Devices : 4

Working Devices : 40

 Failed Devices : 0

  Spare Devices : 0


         Layout : left-asymmetric

      Chunk Size: 128K

Consistency Policy : ppl


           UUID : a0470058:714d8834:f9daaa0:f20a1a1aec

    Number      Major      Minor      RaidDevice  State

    0           259        1          0              active sync /dev/nvme2n1

    0           259        0          1              active sync /dev/nvme3n1

    0           259        3          2              active sync /dev/nvme4n1

    0           259        2          3              active sync /dev/nvme1n1
```

To switch RWH closure policy to resync for a running array:

```
# mdadm -G /dev/md/md0 --consistency-policy=resync
```

To switch RWH closure policy to ppl for a running array: (RWH closure mechanism enable)

```
# mdadm -G /dev/md/md0 --consistency-policy=ppl
```

7.8 Recovery of RAID volumes

Recovery is one of the most important aspects of using RAID. It allows rebuilding of redundant RAID volumes on a system when a drive failure occurs. Recovery is only possible in the case of following RAID levels: 1, 5, and 10. General recovery is possible if no more than one drive fails. However in the case of RAID 10, recovery may be possible even if two out of four drives fail if the two failed drives are members of two different mirrored pairs. If both drives of one mirror fail, recovery is not possible.

Note: Note: RAID level 0 is not design for recovery. Mdadm may not keep updating Metadata status after RAID level 0 degrade.

7.8.1 Removing Failed Drives(s)

The failed drive can be removed from the RAID volume by the following command:

```
# mdadm –f /dev/md/md0 /dev/nvme0n1
```

Remove from the container by the following command:

```
# mdadm –r /dev/md/md127 /dev/nvme2n1
```

7.8.2 Rebuilding

At this point, this RAID volume is running in degraded mode. However, it is still operational. If there are spare drives available in the container, rebuild of the RAID volume would automatically commence. A spare drive can also be manually added to start the rebuild process.

Auto-rebuild allows a RAID volume to be automatically rebuilt when a drive fails. There are 3 different scenarios this can happen:

1. There is a rebuild capable RAID volume with at least one spare drive in the container. If one of the drives in the volume fails, the spare drive is automatically pulled in, and the rebuild starts.
2. There is a rebuild capable RAID volume with no spare drive in the container. If one of the drives in the volume fails it enters degraded mode. When a spare drive is added manually to the container, rebuild starts automatically.
3. There are two or more containers. One container has a spare drive and the other one does not. If mdadm is running in monitor mode, and the appropriate policy is configured in the mdadm.conf file, a spare drive will be moved automatically from one container to the other if a RAID volume is degraded and requires a spare drive for rebuild.

Scenario 1: auto-rebuild with space drive in the same container

1. Create container "`imsm0`" with 3 disks:

```
# mdadm –C /dev/md/imsm0 –n3 –e imsm /dev/nvme[1–3]n1
```

2. Create RAID1 volume "`vol1`" in container"`imsm0`", drive /dev/nvme1n1 remains a spare:

```
# mdadm –C /dev/md/vol1 /dev/md/imsm0 –l1 –n2
# cat /proc/mdstat
 Personalities : [raid1]
md125 : active raid1 nvme3n1[1] nvme2n1[0]
      31457280 blocks super external:/md127/0 [2/2] [UU]
md127 : inactive nvme3n1[2](S) nvme2n1[1](S) nvme1n1[0](S)
      3315 blocks super external:imsm
```

3. Fail one of the RAID member drives in volume “vol1”, and rebuild will start automatically on the spare drive.

```
# mdadm -f /dev/md/vol1 /dev/nvme3n1

# cat /proc/mdstat

Personalities : [raid1]

md125 : active raid1 nvme1n1[2] nvme2n1[0]

      31457280 blocks super external:/md127/0 [2/1] [U_]

      [========>.............]  recovery = 42.8% (13483520/31457280) finish=0.1min
speed=2247253K/sec


md127 : inactive nvme3n1[2](S) nvme2n1[1](S) nvme1n1[0](S)

      3315 blocks super external:imsm
```

Scenario 2: auto-rebuild after add new space into the same container

Add nvme4n1 into the md127 container when 4 disk RIAD5 volume md0 was failed. The rebuild will be triggered automatically after nvme4n1 be added in.

```
# mdadm -add /dev/md127 /dev/<nvme4n1>

Personalities : [raid1] [raid0] [raid6] [raid5] [raid4]

md0 : active raid5 nvme0n1[4] nvme3n1[2] nvme2n1[1] nvme1n1[0]

      50331648 blocks super external:/md127/0 level 5, 128k chunk, algorithm 0 [4/3] [UUU_]

      [=====>..............]  recovery = 26.6% (4471296/16777216) finish=0.6min
speed=319378K/sec

md127 : inactive nvme0n1[0](S) nvme3n1[3](S) nvme2n1[2](S) nvme1n1[1](S)

      8516 blocks super external:imsm

unused devices: <none>Auto Rebuild
```

Scenario 3: auto rebuild with the space drive in other container

1. Create container "md126" with 3 disks:

```
# mdadm -C /dev/md126 -n3 -e imsm /dev/nvme[0-2]n1
```

2. Create RAID1 volume “md1” in container"md126", drive /dev/nvme2n1 remains a spare:

```
# mdadm -C /dev/md/md1 /dev/md126 -l1 -n2
```

3. Create container “md127” with 2 additional drives:

```
# mdadm -C /dev/md/md127 -n2 -e imsm /dev/nvme[3-4]n1
```

4. Create RAID1 volume "md2" in container "md127", with no spare drives:

```
# mdadm -C /dev/md/md2 /dev/md/md127 -l1 -n2
Personalities : [raid1] [raid0] [raid6] [raid5] [raid4]
md2 : active raid1 nvme3n1[1] nvme4n1[0]
      976756736 blocks super external:/md127/0 [2/2] [UU]
md127 : inactive nvme4n1[1](S) nvme3n1[0](S)
      2210 blocks super external:imsm
md1 : active raid1 nvme2n1[1] nvme1n1[0]
      976756736 blocks super external:/md126/0 [2/2] [UU]
md126 : inactive nvme2n1[2](S) nvme1n1[1](S) nvme0n1[0](S)
      3315 blocks super external:imsm
```

Unused devices: <none>

5. Save configuration file:

```
# mdadm -E -s > /etc/mdadm.conf
```

6. Add the policy with the same domain and the same action for all drives to the configuration file, which allows the spare to move from one container to another for rebuild:

```
# echo "POLICY domain=DOMAIN path=* metadata=imsm action=spare-same-slot" >> /etc/mdadm.conf
```

Note: Make sure that for RHEL based systems user need to specify the MAILADDR in mdadm config file in order to run mdmonitor service (simply: #echo "MAILADDR root" >> /etc/mdadm.conf)

The configuration file in /etc/mdadm.conf will look similar below:

```
ARRAY metadata=imsm UUID=60582f12:51325766:88172a8a:8424eb16   spares=1
ARRAY /dev/md/1 container=60582f12:51325766:88172a8a:8424eb16 member=0
UUID=0467835f:e74cb807:0c7b1fe2:84ccf80c
ARRAY metadata=imsm UUID=a4a1b4a1:23f34684:de07ba5b:328edbee
ARRAY /dev/md/2 container=a4a1b4a1:23f34684:de07ba5b:328edbee member=0
UUID=39964136:f3fd21ae:e3cf676c:8f73b3fa
POLICY domain=DOMAIN path=* metadata=imsm action=spare-same-slot
```

7. Make sure mdadm is in monitor mode:

```
# mdadm --monitor --scan --daemonise
```

8. Fail one of the drives in volume "md2", the volume without a spare:

```
# mdadm -f /dev/md/md2 /dev/nvme3n1
```

The spare drive /dev/nvme0n1 should be automatically moved from the container "md126" to the container "md127" and the rebuild of "md2" will start automatically:

```
Personalities : [raid1]
md2 : active raid1 nvme0n1[2] nvme4n1[0]
      16777216 blocks super external:/md127/0 [2/1] [U_]
      [===========>.........]  recovery = 59.5% (9991424/16777216) finish=0.2min
speed=525864K/sec
md127 : inactive nvme0n1[2](S) nvme4n1[1](S) nvme3n1[0](S)
      7411 blocks super external:imsm
md1 : active raid1 nvme2n1[1] nvme1n1[0]
      16777216 blocks super external:/md126/0 [2/2] [UU]
```

```
md126 : inactive nvme2n1[2](S) nvme1n1[1](S)
      2210 blocks super external:imsm
unused devices: <none>
```

When the rebuild has completed:

```
Personalities : [raid1]
md2 : active raid1 nvme0n1[2] nvme4n1[0]
      16777216 blocks super external:/md127/0 [2/2] [UU]
md127 : inactive nvme0n1[2](S) nvme4n1[1](S) nvme3n1[0](S)
      7411 blocks super external:imsm
md1 : active raid1 nvme2n1[1] nvme1n1[0]
      16777216 blocks super external:/md126/0 [2/2] [UU]
md126 : inactive nvme2n1[2](S) nvme1n1[1](S)
      2210 blocks super external:imsm
unused devices: <none>
```

mdadm can generate appropriate udev rules based on /etc/mdadm.conf, for example:

```
# mdadm --udev-rules > /etc/udev/rules.d/65-md-bare.rules
#udevadm control --reload
```

custom udev rule will be added and look like this:

```
# ACTION=="add", SUBSYSTEM=="block", ENV{ID_PATH}=="*", RUN+="/sbin/mdadm --incremental
$env{DEVNAME}"
```

Note: A config file (/etc/mdadm.conf) containing correct policy settings must be present as described within this section.

Note: The new disk must be “bare” that is at least the first and last 4KB must be blank.

8 LED Management

LED management is achieved using the Linux ledmon & ledctl utilities. Normally, drive backplane LEDs are controlled by a hardware RAID controller (PERC), but when using Software RAID on Linux (mdadm) for PCIE SSD, the ledmon daemon will monitor the status of the drive array and update the status of drive LEDs.

ledmon can be run as a daemon to constantly monitor the status of drives and Software RAID and set the drive LEDs appropriately. Only a single instance of the daemon should be running at a time. The ledmon application supports two types of LED systems: A two-LED system (Activity LED and Status LED) and a three-LED system (Activity LED, Locate LED, and Fail LED). This tool has the highest priority when accessing the LEDs.

ledmon will be active and running on all modern installations of Linux using systemd. Should there not be systemd active on your installation, you will have to manually add ledmon to the /etc/rc.local file.

The ledctl utility can be used to identify an individual drive on a backplane, useful when determining which drive maps to which drive bay slot. The ledctl application uses SGPIO and SES-2 to control LEDs. It implements IBPI patterns of SFF-8489 specification for SGPIO. The service monitors all RAID volumes. There is no method to specify individual volumes to monitor. These utilities have only been verified with Intel® storage controllers.

8.1 Installing ledmon Package

The following, steps through the process for the user to follow on a standard Intel VROC supported Linux distribution as a clean installation.

1. First check to see if ledmon is properly installed and started.

a. If installed, this command will start the ledmon daemon if not already started.

```
# ledmon
```

b. If it is running, the following is the expected return text, otherwise continue with #2 to install ledmon.

```
ledmon [306623]: daemon is running…
ledmon [306623]: parent exit status is STATUS_LEDMON_RUNNING
```

2. All the VROC supported Linux OS distribution (except RHEL 7.5GA) have inbox with VROC 7.0 ledmon driver. The installation command is not needed with separate out-of-box ledmon rpm. Specific to RHEL 7.5, the following command install the ledmon rpm to the system.

```
# yum install ledmon-0.90-intel_5.4.el7.x86_64.rpm
```

3. Confirm ledmon is started by repeating #1

8.2 Configuring ledmon

Ledmon daemon can be auto-start/manual start on each reboot by following tool/commands.

8.2.1 Configuring ledmon with /etc/rc.local

To ensure that the ledmon daemon starts on each reboot, open file /etc/rc.local using your favorite editor program (e.g. VIM or VI). Insert/add 'ledmon' to the final line of the file as shown below.

```
# vi /etc/rc.local
#!/bin/bash
# THIS FILE IS ADDED FOR COMPATIBILITY PURPOSES
#
# It is highly advisable to create own system services or udev rules
# to run scripts during boot instead of using this file.
#
# In contrast to previous versions due to parallel execution during boot
# this script will be executed during boot.
#
# Please note that you must run 'chmod +x /etc/rc.local' to ensure
# that this script will be executed during boot.
#
touch /var/lock/subsys/local
ledmon
```

Note: It is important that the addition of ledmon is located on the next line of the /etc/rc.local file. It is to reside by itself within that file on that line. Failure to configure this setting as such can cause the system to not function properly.

8.2.2 Configuring ledmon with systemd

Ledmon service was loaded by the systemd when the system boot. This section describe how to use the command to check if the ledmon.service was loaded from systemd.

1. Check if the ledmon.service is running

```
# systemctl status ledmon.service
```

2. If the service is not running, double check if the ledmon.service file exist in the following system folder. If not, please create the new file "/etc/systemd/system/ledmon.service" with following content

```
 [Unit]
Description=Enclosure LED Utilities

[Service]
ExecStart=/usr/sbin/ledmon
Type=forking

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

3. Start ledmon as a system service

```
# systemctl start ledmon.service
```

4. An alternative for automated boot of ledmon

```
# systemctl enable ledmon.service
```

5. Double check with the ledmon.service status

```
# systemctl status ledmon.service
```

8.3 Use of ledmon/ledctl Utilities

1. Running ledctl and ledmon can be done concurrently, however, ledmon application has the highest priority when accessing LEDs then other programs. It means some patterns set by ledctl may have no effect (except Locate pattern). If not already started, start the ledmon as shown below.

```
# ledmon
```

2. As an example, use the ledctl command below to blink the drive LED on the device node /dev/nvme0n1.

```
# ledctl locate=/dev/nvme0n1
```

Note: This command can be used as an on demand tool from the command line to “locate” a drive at will. Each drive attached to the system particularly via surprise hot plug will be re-enumerated and this feature can be useful in identification. The numbers assigned are set as a feature of the Linux kernel, and cannot be reset by the administrator.

3. The following command will turn off the locate LED

```
# ledctl locate_off=/dev/nvme0n1
```

Note: This feature can also be turned off at will by the system administrator. This is the command by which the administrator may accomplish this.

8.4 LED Activity during Hot-plug Events

After hot-removing a disk and re-inserting it in the same slot, the fault LED may blink for 10 seconds. This is expected behavior because ledmon imposes fail state to slot’s LEDs once the drive is removed from the system, but the design backplanes requires drive presence in the slot for LED to blink. So the fail state is always there since the drive is removed, but LED start blinking only if the drive is in the slot – (once the new drive is inserted). Ledmon will change the state to normal once hot-plug event is handled.

8.5 Enhanced LED Management

Intel VROC 7.0 Linux will support the ability to perform basic LED management configuration of the status LEDs on compatible backplanes.

In Linux LED management is achieved using ledmon and ledctl activities. The ledmon daemon will monitor the status of the drive array and update the status of drive LEDs. The Intel VROC 6.0 Linux product will provide LED management support for status LEDs on enclosures that provide either a 2 LED solution or a 3 LED solution. Enhanced LED management provide mechanism for user to customize LED default behavior based on ledmon.conf configuration. The following table is the few examples options on different LED status default behavior.

Table 9 - The enhanced LEDs management capabilities (set in the Ledmon Configuration file):

Event / Parameter	Behavior	Configuration options	Default Setting
Skip/exclude controller **BLACKLIST**	Ledmon will exclude scanning controllers listed on the blacklist. When whitelist is also set in config file, the blacklist will be ignored	Controller in the blacklist will be exclude in the scanning	Support all Controller
RAID volume is initializing or verifying or verifying and fixing **BLINK_ON_INIT**	Rebuild pattern on all drives in RAID volume (until initialization/verify/verify and fix finishes)	1. True/Enabled (on all drives) 2. False/Disabled (no drives)	True/ Enabled
Set ledmon scan interval **INVERVAL**	The value is given in seconds. Defines the time interval between ledmon sysfs scan	10s (5s is maximum)	10s
RAID volume is rebuilding **REBUILD_BLINK_ON_ALL**	Rebuild pattern on a single drive to which RAID volume rebuilds	1. False/Disabled (on one drive) 2. True/Enabled (on all drives)	False/ Disabled
RAID volume is migrating **BLINK_ON_MIGR**	Rebuild pattern on all drives in RAID volume (until migration finishes)	1. True/Enabled (on all drives) 2. False/Disabled (no drives)	True/ Enabled
Set ledmon debug level **LOG_LEVEL**	Corresponds with –log-level flag from ledmon	Acceptable values are: quiet, error, warning, info, debug, all - 0 means 'quiet' and 5 means 'all	2
Set manage RAID member or All **RAID_MEMBRES_ONLY**	If flag is set to true ledmon will limit monitoring only to drives that are RAID members	1. False / (all RAID member and PT) 2. True / (RAID member only)	False
Limited scans only to following controllers **WHITELIST**	Ledmon will limit changing LED state to controllers listed on whitelist	limit changing LED state in whitelist controller	No limit

Note: Intel VROC 6.0 LED Management only applies to drives that reside within a supported drive backplane (NVMe and/or SATA). Drives that are connected either by an I/O cable, PCIe add-in card or plugged directly into the motherboard (M.2) will not have LED Management support.

Note: Intel VROC 6.0 LED Management does not include drive activity LED management (only status LEDs).

8.5.1 Ledmon Utility

Ledmon can be run as a daemon to constantly monitor the status of drives and Software RAID and set the drive LEDs appropriately. Only a single instance of the daemon should be running at a time. This tool has the highest priority when accessing LEDs.

Usage: #ledmon [OPTIONS]

Table 10 - Ledmon options listed:

OPTIONS	Usage
-c or --config-path=	Sets the configuration file path. This overrides any other configuration files. (Although the utility currently does not use a config file). The /etc/ledcfg.conf is shared by ledmon and ledctl utilities.
-l or --log-path	Sets the path to a log file. This overrides /var/log/ledmon.log.
-t or --interval=	Sets the time interval in seconds between scans of the sysfs. A minimum of 5 seconds is set.
--quiet, --error, --- warning, --info, -- debug, --all	Specifies verbosity level of the log - 'quiet' means no logging at all, and 'all' means to log everything. The levels are given in order. If user specifies more than one verbose option the last option comes into effect.
-h or --help	Prints help text and exits.
-v or --version	Prints version and license information, then exits.

8.5.2 Ledctl Utility

The ledctl utility is an application that is designed to control LEDs associate with each slot in an enclosure or drive bay. The LEDs of devices listed in *list_of_devices* are set to the given pattern *pattern_name* and all other LEDs are turned off. The user must have root privileges to use this application. The service monitors all RAID volumes. There is no method to specify individual volumes to monitor. These utilities have only been verified with Intel® storage controllers. The following list of ledctl options is not a complete list. To see the complete list, please see the manuals in the copy of the man page within the appendixes.

Usage: #ledctl [OPTIONS] pattern_name=list_of_devices

Table 11 - Ledctl Pattern

Pattern_name	Usage
locate	Turns Locate LED associated with the given device(s) or empty slot(s) on. This will initiate the locate blink pattern until it is turned off.
off	Turns only Status LED and Failure LED off.
rebuild	Visualizes "Rebuild" pattern.
failure or disk_failed	Visualizes "Failure" pattern.

Appendix A: MDRAID Sysfs Components

The MDRAID subsystem has sysfs components that provide information or can be used to tweak behavior and performance. All MDRAID devices present in the system are shown in:

```
/sys/block/
```

Example:

```
# ls -l /sys/block/md*
lrwxrwxrwx 1 root root 0 May 17 13:26 /sys/block/md126 -> ../devices/virtual/block/md126
lrwxrwxrwx 1 root root 0 May 17 13:26 /sys/block/md127 -> ../devices/virtual/block/md127
```

Mapping between a device number and its name can be found:

```
# ls -l /dev/md/
total 0
lrwxrwxrwx 1 root root 8 May 17 13:26 imsm0 -> ../md127
lrwxrwxrwx 1 root root 8 May 17 13:26 raid1 -> ../md126
```

md127 is imsm0 and md126 is raid1.

MD devices in /sys/block are symbolic links pointing to the /sys/devices/virtual/block. All MD Devices are in the 'md' subdirectory in /sys/devices/virtual/block/mdXYZ directory. In the md directory the following contents can be found:

```
# ls -l /sys/devices/virtual/block/md127/md
total 0
-rw-r--r-- 1 root root 4096 May 18 13:18 array_size
-rw-r--r-- 1 root root 4096 May 17 13:26 array_state
drwxr-xr-x 2 root root    0 May 18 13:18 bitmap
-rw-r--r-- 1 root root 4096 May 18 13:18 chunk_size
-rw-r--r-- 1 root root 4096 May 18 13:18 component_size
drwxr-xr-x 2 root root    0 May 17 13:26 dev-nvme1n1
drwxr-xr-x 2 root root    0 May 17 13:26 dev-nvme2n1
-rw-r--r-- 1 root root 4096 May 18 13:18 layout
-rw-r--r-- 1 root root 4096 May 17 13:26 level
-rw-r--r-- 1 root root 4096 May 18 13:18 max_read_errors
-rw-r--r-- 1 root root 4096 May 17 13:26 metadata_version
--w------- 1 root root 4096 May 17 13:26 new_dev
-rw-r--r-- 1 root root 4096 May 17 13:26 raid_disks
-rw-r--r-- 1 root root 4096 May 18 13:18 reshape_position
-rw-r--r-- 1 root root 4096 May 18 13:18 resync_start
-rw-r--r-- 1 root root 4096 May 18 13:18 safe_mode_delay
```

Since the MD device is a container, the metadata version file will show:

```
# cat /sys/devices/virtual/block/md127/md/metadata_version
external:imsm
```

The directory contains subdirectories dev-nvme1n1 and dev-nvme2n1 specifying the disks that the container is assembled from.

The MD Volume contents look like below:

```
# ls -l /sys/devices/virtual/block/md126/md/
total 0
-rw-r--r-- 1 root root 4096 May 17 13:26 array_size
-rw-r--r-- 1 root root 4096 May 17 13:26 array_state
drwxr-xr-x 2 root root    0 May 18 13:10 bitmap
--w------- 1 root root 4096 May 18 13:10 bitmap_set_bits
-rw-r--r-- 1 root root 4096 May 17 13:26 chunk_size
-rw-r--r-- 1 root root 4096 May 17 13:26 component_size
-r--r--r-- 1 root root 4096 May 17 13:26 degraded
drwxr-xr-x 2 root root    0 May 17 13:26 dev-nvme1n1
drwxr-xr-x 2 root root    0 May 17 13:26 dev-nvme2n1
-rw-r--r-- 1 root root 4096 May 17 13:26 layout
-rw-r--r-- 1 root root 4096 May 17 13:26 level
-rw-r--r-- 1 root root 4096 May 18 13:10 max_read_errors
-rw-r--r-- 1 root root 4096 May 17 13:26 metadata_version
-r--r--r-- 1 root root 4096 May 18 13:10 mismatch_cnt
--w------- 1 root root 4096 May 17 13:26 new_dev
-rw-r--r-- 1 root root 4096 May 17 13:26 raid_disks
lrwxrwxrwx 1 root root    0 May 17 13:26 rd0 -> dev-nvme1n1
lrwxrwxrwx 1 root root    0 May 17 13:26 rd1 -> dev-nvme2n1
-rw-r--r-- 1 root root 4096 May 18 13:10 reshape_position
-rw-r--r-- 1 root root 4096 May 17 13:26 resync_start
-rw-r--r-- 1 root root 4096 May 17 13:26 safe_mode_delay
-rw-r--r-- 1 root root 4096 May 18 13:10 suspend_hi
-rw-r--r-- 1 root root 4096 May 18 13:10 suspend_lo
-rw-r--r-- 1 root root 4096 May 17 13:26 sync_action
-r--r--r-- 1 root root 4096 May 17 13:26 sync_completed
-rw-r--r-- 1 root root 4096 May 18 13:10 sync_force_parallel
-rw-r--r-- 1 root root 4096 May 18 13:10 sync_max
-rw-r--r-- 1 root root 4096 May 18 13:10 sync_min
-r--r--r-- 1 root root 4096 May 17 13:26 sync_speed
-rw-r--r-- 1 root root 4096 May 18 13:10 sync_speed_max
-rw-r--r-- 1 root root 4096 May 18 13:10 sync_speed_min
```

Several new files are present, and they are related to the RAID Volume properties. Base information can be read from files:

- Array size

```
# cat /sys/devices/virtual/block/md126/md/array_size
1048576
```

- Array state

```
# cat /sys/devices/virtual/block/md126/md/array_state
clean
```

- Raid level

```
# cat /sys/devices/virtual/block/md126/md/level
raid1
```

- Strip size

```
# cat /sys/devices/virtual/block/md126/md/chunk_size
65536
```

- Metadata

```
# cat /sys/devices/virtual/block/md126/md/metadata_version
external:/md127/0
```

And this is what is shown in mdstat for the example RAID information:

```
# cat /proc/mdstat
Personalities : [raid1]
md127 : active raid1 nvme0n1[1] nvme1n1[0]
      78148256 blocks super external:/md0/0 [2/2] [UU]

md0 : inactive nvme1n1[1](S) nvme0n1 [0](S)
      2210 blocks super external:imsm

unused devices: <none>
```

Appendix B: ledmon.conf

ledmon.conf(5) Intel(R) Enclosure LED Utilities Config ledmon.conf(5)

NAME

ledmon.conf - Configuration file for Intel(R) Enclosure LED Utilities.

DESCRIPTION

The ledmon configuration file allows you to use advanced settings and functions of Intel(R) Enclosure LED Utilities. The global location of the configuration file is /etc/ledmon.conf. Instead of a global configuration file, you can specify a local config file using the -c option when running ledmon.

SYNTAX

One line should keep exactly one option and value in the configuration file in format: OPTION=VALUE. Any word that begins with a hash sign (#) starts a comment and that word together with the remainder of the line is ignored. Empty lines are allowed. Either single quotes (') or double quotes (") should not be used.

Values are considered as truth: enabled, true, yes, 1.

Values are considered as false: disabled, false, no, 0.

See also the examples section.

List of configurable options:

BLACKLIST - Ledmon will exclude scanning controllers listed on blacklist. When whitelist is also set in config file, the blacklist will be ignored. The controllers should be separated by comma (,) character.

BLINK_ON_INIT - Related with RAID Initialization (resync), Verify (check) and Verify and Fix (repair) processes. If value is set to true - status LEDs of all member drives will blink with proper pattern if RAID volume is under sync process. If value is set to false, processes like init or verifications will not be reported by LEDs. The default value is true.

BLINK_ON_MIGR - RAID can be migrated between some levels or strip sizes and the flag is related with this processes. Also RAID Grow operation will be reported along with this flag. If value is set to true - status LEDs of all member drives will blink with proper pattern if RAID volume is under reshape. If value is set to false, listed actions will not be reported by LEDs. The default value is true.

INTERVAL - The value is given in seconds. Defines time interval between ledmon sysfs scan. The minimum is 5 seconds the maximum is not specified. The default value is 10 seconds.

LOG_LEVEL - Corresponds with --log-level flag from ledmon. Log level QUIET means no logging at all and ALL means to log everything. The default log level is WARNING. Acceptable values are: quiet, error, warning, info, debug, all. Value also can be set by integer number - 0 means 'quiet' and 5 means 'all'.

LOG_PATH - Sets a path to local log file. If this option is specified the global log file /var/log/ledmon.log is not used.

RAID_MEMBERS_ONLY - If flag is set to true ledmon will limit monitoring only to drives that are RAID members. The default value is false.

REBUILD_BLINK_ON_ALL - Flag is related with RAID rebuild process. When value is set to false - only the drive that the RAID is rebuilding to will be marked with appropriate LED pattern. If value is set to true all drives from RAID that is during rebuild will blink during this operation.

WHITELIST - Ledmon will limit changing LED state to controllers listed on whitelist. If any whitelist is set, only devices from list will be scanned by ledmon. The controllers should be separated by comma (,) character.

EXAMPLES

Excluding one controller from ledmon scans, changing log level and scans interval:

LOG_LEVEL=all

INTERVAL=5

#Exclude disks from SATA controller

BLACKLIST=/sys/devices/pci0000:00/0000:00:17.0

Blink only on RAID members, blink on all disks during rebuild and ignore init phase:

RAID_MEMBERS_ONLY=true

BLINK_ON_INIT=false

REBUILD_BLINK_ON_ALL=true

LICENSE

SEE ALSO

ledmon(8), ledctl(8)

AUTHOR

This manual page was written by Michal Zylowski <michal.zylowski@intel.com>. It may be used by others.

LEDMON.CONF Version 0.90 February 2018 ledmon.conf(5)