# Intel® Virtual RAID on CPU (Intel® VROC) and Linux* Performance Testing (using FIO*)

Performance Evaluation Guide

*August 2021*

Document Number: 337095-005US

| Revision Number | Description | Revision Date |
|---|---|---|
| 001 | • Initial release | February 2018 |
| 002 | • Updated and reworked document to align with current best known methods (BKMs) and processes for testing performance | January 2019 |
| 003 | • Added link to FIO Script Reference Code | May 2019 |
| 004 | • Fixed hyperlinks<br>• Moved FIO Download & Installation details | June 2019 |
| 005 | • Updated whole document with more recent data, new logo, new software, etc.<br>• Revised it so it is no longer specific to a particular SSD | August 2021 |

# Legal Disclaimer

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. – Current characterized errata are available on request.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors.

Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit www.intel.com/benchmarks.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available security updates. See below for configuration details. No product or component can be absolutely secure.

Testing performed by Intel on August 5, 2021. System configuration:

Intel® Server Board M50CYP Family, BIOS Version: SE5C620.86B.01.01.0003.2104260124, BIOS Release Date: 04/26/2021, Intel® VROC Pre-OS Version: 7.5.0.1152
CPUs: 2 x Intel® Xeon® Platinum 8358 CPU @ 2.60GHz (32 cores each)
RAM: 128GB RAM (16 x 8GB Micron* DDR4 3200MT/s SDRAM RDIMMs)
OS: Red Hat* Enterprise Linux Server 8.1, Kernel: 4.18.0-147.el8.x86_64
mdadm version: mdadm - v4.1 - 2018-10-01, kmod-iavmd-1.0.0.1494-rhel_81.x86_64.rpm package installed
Added "initcall_blacklist=vmd_drv_init pci=pcie_bus_perf" to grub boot option which disables inbox VMD, loads iavmd, and maximizes MaxPayload for each NVMe device
4 x 3.84TB Intel® D7-P5510 Series PCIe 4.0 U.2 SSDs (Model: SSDPF2KX038TZ, Firmware: JCV10016)

Your costs and results may vary.

Intel technologies may require enabled hardware, software, or service activation.

# Contents

# 1 Setup

This is a quick start guide to using fio* benchmarking tool for systems containing Intel® Xeon® Scalable Processors with Red Hat* Enterprise Linux. This document includes testing of the Intel® Solid State Drive DC P5510 Series with specific steps for testing NVMe SSD RAID arrays on Intel® Volume Management Device (VMD) - enabled PCIe lanes.

## 1.1 Hardware configuration

For maximum use of Intel® Virtual RAID on CPU (VROC) with availability to RAID 0, RAID 5, RAID 1, and RAID 10, a VROC hardware key ("Premium" or "Intel-SSD-only") must be installed on the motherboard.

On an Intel® Coyote Pass server board M50CYP2SB2U (Chassis: M50CYP2UR208BPP), there are eight NVMe U.2 form factor drive bay slots (four are on the 1st CPU and four are on the 2nd CPU). Make sure to connect the drives that will be used for testing on the same CPU. Performance becomes degraded when drives are spanning CPUs. Your setup may be different. Please check with your OEM for details on how to connect your drives so they are not spanning CPUs.

It is recommended to populate all DIMM channels with RAM from the same manufacturer, same model, same size, same speed, etc.

## 1.2 BIOS settings

The following BIOS settings are based on an Intel® Coyote Pass server board M50CYP and are recommended for maximum performance. Please refer to the instructions that have been supplied by your OEM as those instructions may differ from the set below.

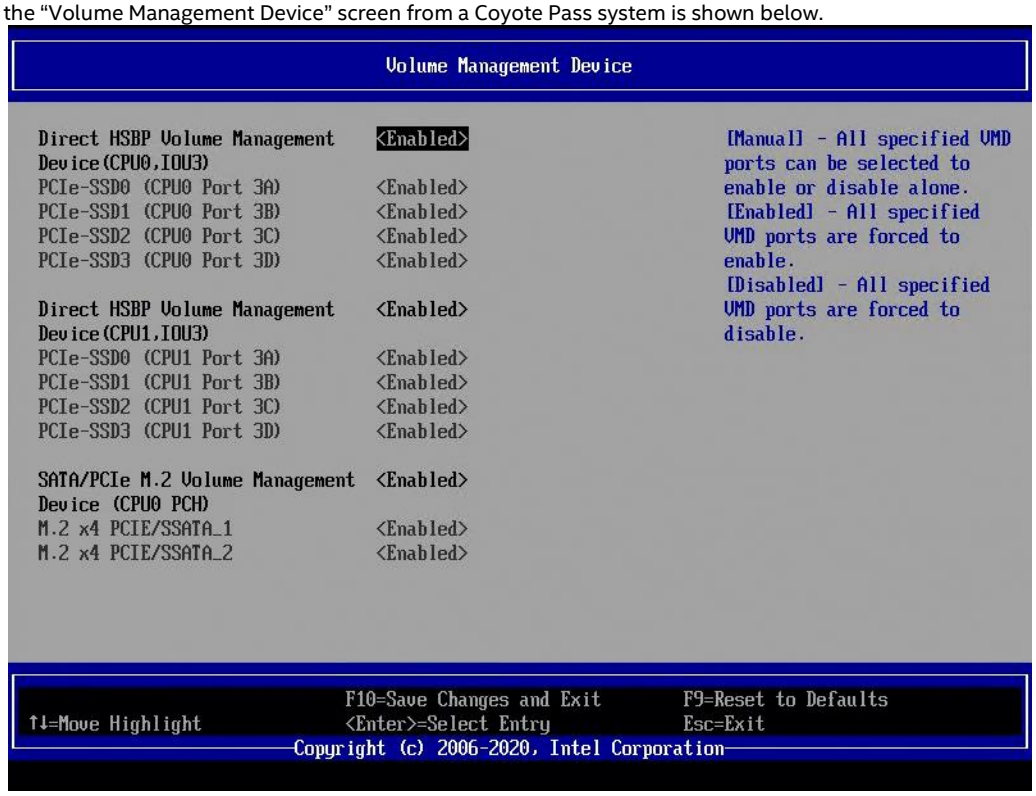
Enter the BIOS at boot up by pressing <F2>. Load the defaults by pressing <F9>. Press <F10> to save and reboot the system for the changes to take effect. Enter BIOS again by pressing <F2>. Once you are in the BIOS, navigate to the following...

Verify... Advanced → Processor Configuration → Intel® Hyper-Threading Tech → Enabled
Change... Advanced → Power & Performance → CPU Power and Performance Policy → Performance
Change... Advanced → Power & Performance → Workload Configuration → I/O Sensitive
Verify... Advanced → Power & Performance → Uncore Power Management → Performance P-limit → Enabled
Verify... Advanced → Power & Performance → CPU P State Control → Enhanced Intel SpeedStep® Tech → Enabled
Verify... Advanced → Power & Performance → CPU P State Control → Intel® Turbo Boost Technology → Enabled
Verify... Advanced → Power & Performance → CPU P State Control → Energy Efficient Turbo → Enabled
Verify... Advanced → Power & Performance → Hardware P States → Hardware P-States → Native Mode
Verify... Advanced → Power & Performance → Hardware P States → HardwarePM Interrupt → Disabled
Verify... Advanced → Power & Performance → Hardware P States → EPP Enable → Enabled
Verify... Advanced → Power & Performance → Hardware P States → APS rocketing → Disabled
Verify... Advanced → Power & Performance → Hardware P States → Scalability → Disabled
Verify... Advanced → Power & Performance → Hardware P States → RAPL Prioritization → Disabled
Change... Advanced → Power & Performance → CPU C State Control → Package C-State → C0/C1 State
Verify... Advanced → Power & Performance → CPU C State Control → C1E → Disabled
Verify... Advanced → Power & Performance → CPU C State Control → Processor C6 → Enabled
Change... Advanced → System Acoustic and Performance Configuration → Set Fan Profile → Performance
Change... Advanced → System Acoustic and Performance Configuration → Fan PWM Offset → 65

With Intel VROC, Intel VMD will need to be enabled on any ports containing NVMe SSDs on your platform. The BIOS settings below apply to a Coyote Pass system that has eight NVMe U.2 drive bay slots connected to the backplane (4 are on CPU0 and 4 are on CPU1) and two onboard M.2 slots on the motherboard. Your setup may be different than below. Please check with your OEM if you are unsure where to enable VMD on your platform.

Change... Advanced → Integrated IO Configuration → Volume Management Device → Direct HSBP Volume Management Device (CPU0,IOU3) → Enabled
Change... Advanced → Integrated IO Configuration → Volume Management Device → Direct HSBP Volume Management Device (CPU1,IOU3) → Enabled
Change... Advanced → Integrated IO Configuration → Volume Management Device → SATA/PCIe M.2 Volume Management Device (CPU0 PCH) → Enabled

A screenshot of the "Volume Management Device" screen from a Coyote Pass system is shown below.




## 1.3 Intel® Virtual RAID on CPU (from the BIOS)

The following info is based on an Intel® Coyote Pass server board M50CYP2UR. For other platforms, accessing Intel Virtual RAID on CPU in the BIOS may differ from below.

After setting the BIOS settings from the "BIOS settings" section above, reboot the system, and enter BIOS again by pressing <F2>.

Navigate to Advanced → PCI Configuration → UEFI Option ROM Control → Intel® Virtual RAID on CPU

From here, you will be able to see the VROC hardware key installed, Intel® VROC Pre-OS version number, and any existing VROC RAID volumes. Select "All Intel VMD Controllers" to view the applicable NVMe SSDs.

## 1.4 Red Hat Enterprise Linux installation/dependencies

All Red Hat Enterprise Linux installations shall be in UEFI mode.

After entering the "INSTALLATION SUMMARY" screen, the "Minimal Install" option under "SOFTWARE SELECTION" is sufficient enough for this performance testing.

Once installation is complete, setup a repository containing all the necessary rpm packages and install the following dependencies by running…

```
yum install gcc libaio-devel zlib-devel unzip sysstat libreport-filesystem numactl
    sg3_utils nvme-cli pciutils mdadm
```

With newer platforms, it is recommended to increase Max Payload Size (MPS) on any NVMe devices connected to the system for best performance. To do this, add "`pci=pcie_bus_perf`" to the grub menu by entering the following and rebooting the system…

```
grubby --args="pci=pcie_bus_perf" --update-kernel ALL
```

It may be beneficial to install the latest kmod-iavmd driver to boost performance. Check with your OEM on obtaining this driver. Once downloaded, the driver can be installed using yum. After installation, verify "`initcall_blacklist=vmd_drv_init`" was added to the grub menu by running "`grubby --info $(grubby --default-kernel)`". This disables the inbox VMD driver so the kmod-iavmd driver can be used. If it isn't in the grub menu, run the following to add it to the grub menu and reboot the system…

```
grubby --args="initcall_blacklist=vmd_drv_init" --update-kernel $(grubby --default-
    kernel)
```

## 1.5 Downloading and installing Intel® Memory and Storage Tool (IntelMAS)

Intel® Memory and Storage Tool (IntelMAS) can be used to update firmware on Intel® SSDs and can also be used to perform a low-level format on Intel® NVMe SSDs. IntelMAS can be downloaded at https://downloadcenter.intel.com/download/30509?v=t. *Note: Intel® Memory and Storage Tool (IntelMAS) replaced Intel® SSD Data Center Tool (ISDCT). The commands for both tools are very similar*. An example of how to install IntelMAS is shown below.

```
unzip Intel®_MAS_CLI_Tool_Linux_1.8.zip
yum install intelmas-1.8.140-0.x86_64.rpm
```

Other useful commands are …

```
# Display version
intelmas version

# Display the usage and any applicable parameter options
intelmas

# List and show basic information about any Intel SSDs
intelmas show -intelssd

# List and show detailed information about any Intel SSDs
intelmas show -all -intelssd

# Update firmware on Intel SSD
intelmas load -intelssd <index of SSD>

# Format Intel NVMe SSD
intelmas start -nvmeformat -intelssd <index of SSD>
```

## 1.6 Downloading and installing FIO benchmarking tool

Download the fio master zip from GITHUB at https://github.com/axboe/fio

You will also need libaio-devel rpm in order to install the necessary .h files to build the libaio ioengine. Also, rpms gcc and zlib-devel are needed for proper fio operation. These rpms should be on the installation source for your Linux distribution. See Red Hat Enterprise Linux installation/dependencies section for installing these dependencies.

```
unzip fio-master.zip
cd fio-master
./configure && make && make install
# Verify fio installed successfully by checking the version
fio --version
```

# 2 Preparing the VROC RAID volume

## 2.1 Formatting the Intel® NVMe SSDs

Drives with previous RAID metadata or drives used for other applications could impact performance. Clearing the metadata and erasing or performing a low-level format can stabilize your drives providing more consistent results.

Remove any metadata from the drives:

```
mdadm --zero-superblock /dev/nvme*n1
```

Show IntelMAS indexes of the applicable Intel SSDs:

```
intelmas show -intelssd
```

Perform a low-level format on each of the applicable Intel SSDs:

```
intelmas start -nvmeformat -intelssd <index of SSD>
```

Repeat this low-level format for all the drives to be used in the new array.

## 2.2 Creating the RAID Array

Intel VROC uses `mdadm` to create and manage RAID volumes when VMD is enabled in the BIOS. The available RAID volumes are RAID 0, RAID 5, RAID 1, and RAID 10.

First start by creating a VROC metadata container. The example below creates a device called /dev/md/imsm0 using NVMe devices /dev/nvme0n1, /dev/nvme1n1, /dev/nvme2n1, and /dev/nvme3n1. The "`-n 4`" parameter means number of disks to use. The "`-e imsm`" parameter defines the RAID metadata style to use. This example uses the Intel® Matrix Storage Manager (IMSM) RAID metadata style.

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

Next, create the RAID volume. The example below creates a 4-disk RAID 0 (4DR0) by creating a device called /dev/md/4DR0 using the container created from above (/dev/md/imsm0). The "`-n 4`" parameter means number of disks used. The "`-l 0`" parameter sets the RAID level to 0. The "`-c 128`" parameter sets the chunk size to 128k. Note: leaving out this parameter will set the chunk size to the default depending on the RAID level.

```
mdadm --create /dev/md/4DR0 /dev/md/imsm0 -n 4 -l 0 -c 128
```

Default chunk sizes:
RAID 0: 128k
RAID 5: 128k
RAID 1: n/a
RAID 10: 128k

**Note:** If no other RAID volumes existed before creating the RAID volume, the block devices will be /dev/md127 for the container and /dev/md126 for the RAID volume.

After creating a RAID5 volume, there is a setting that has demonstrated slightly improved RAID5 performance. This setting is not found with RAID 0, RAID 1, or RAID 10. Apply the following setting to a RAID 5 volume…

```
echo 4 > /sys/block/md126/md/group_thread_cnt
```

**Note**: A group thread count of 0 is default and limited testing has been conducted with other group thread counts. We found a group thread count of 4 to be the most efficient boosting performance about 10%. More testing is planned to determine optimal performance and latency impacts. Larger group thread counts appear to impact read performance.

After creation of RAID 5, RAID 1, or RAID 10 volumes, initialization will start automatically. To speed up initialization or rebuild times, the following setting can be applied. **Note:** the default setting for "`sync_speed_max`" is 200,000.

```
echo 5000000 > /sys/block/md126/md/sync_speed_max
```

## 2.3 Creating XFS filesystem (not required for benchmark testing)

Most benchmark testing is performed on raw devices, however if you would like to see results from a filesystem installed, this section will explain it. Drives perform better when data is aligned. 4k alignment is ideal for most SSDs. The following steps will demonstrate how to create partitions so they are aligned (minimal and optimal) and the steps will show how to create the XFS filesystem. The following example applies to a VROC RAID volume.

```
parted –a optimal –s /dev/md126 mklabel gpt mkpart primary 0% 100%
```

Checking the alignment of the newly created partition:

```
parted /dev/md126 align-check optimal 1
parted /dev/md126 align-check minimal 1
```

Creating the XFS filesystem:

```
mkfs.xfs /dev/md126p1
```

Mounting the filesystem:

```
mkdir –p /mnt/VROC
mount /dev/md126p1 /mnt/VROC
```

**Note:** If you plan on running fio on any XFS filesystem, you must fill the drive with sequential writes first before running random workloads. Running random workloads from fio first on an empty XFS system may generate errors.

**Note:** There are a few parameters that need to be adjusted in the fio script before running tests on the XFS filesystem. First the "`filename=`" parameter will need to reflect the location of the filesystem mount plus the fio filename you plan to use (e.g., "`filename=/mnt/md126p1/iofile`"). If the filesystem does not currently have the fio file (e.g., "iofile") in the directory, then the "`time_based`" parameter must be eliminated in the first fio workload which is the sequential pre-conditioning run as described in the "Conditioning the drive or volume before testing" section of this document. Also, the "`size=`" parameter cannot be set using a percentage like "`100%`" and it cannot be the full size of the volume. An actual size must be specified and it must be at least a few Gigabytes smaller than the full size. For example, if the RAID volume is 16TB in size, the parameter can be "`size=15892G`". After the fio file is created in the directory, all tests should use the "`time_based`" parameter moving forward and the "`size=`" parameter can go back to "`size=100%`".

# 3 Running a performance test

Stopping the IRQBALANCE service before testing is recommended to minimize high standard deviations between iterations. This can be stopped by typing "`systemctl stop irqbalance`". Do not disable IRQBALANCE because it may cause CPU soft lockup errors and kernel panics. If IRQBALANCE was disabled, please re-enable and reboot the system.

Benchmarking a drive, array, or workload should be reproducible and relevant. A short burst test, (e.g., 60 seconds) does not give the storage media enough time to reach steady state, and therefore the results can be unreliable. Also, multiple iterations should be run to verify the reproducibility of results. All testing should be ran for a minimum of 10 minutes (preferably 10 minutes of ramping followed by 10 minutes of run time). The test sequence should be repeated at least 2 more times to get a statistical set of data to take averages and standard deviation. If your results vary greater than +-5%, you may have to check your test parameters, devices under test, and overall system. Additionally, each OS has its own set of performance counters to capture IOPS, bandwidth, and CPU usage (overall). More data is better when attempting to predict real world performance by using synthetic benchmarks.

When testing 128KB sequential READs/WRITEs, with four devices in a RAID 0 or RAID 5, it is recommended to have an IO depth of 128, but the device may not peak until it reaches an IO depth of 256. For 4K random, the `numjobs` (threads or workers) and `iodepth` should be increased to fully saturate the device. If using four devices in a RAID 0 or RAID 5, the `iodepth` 256 may be needed to obtain peak value, depending on the SSD. 16 or more `numjobs` or threads may be needed to identify the right combination of IOPS and latency.

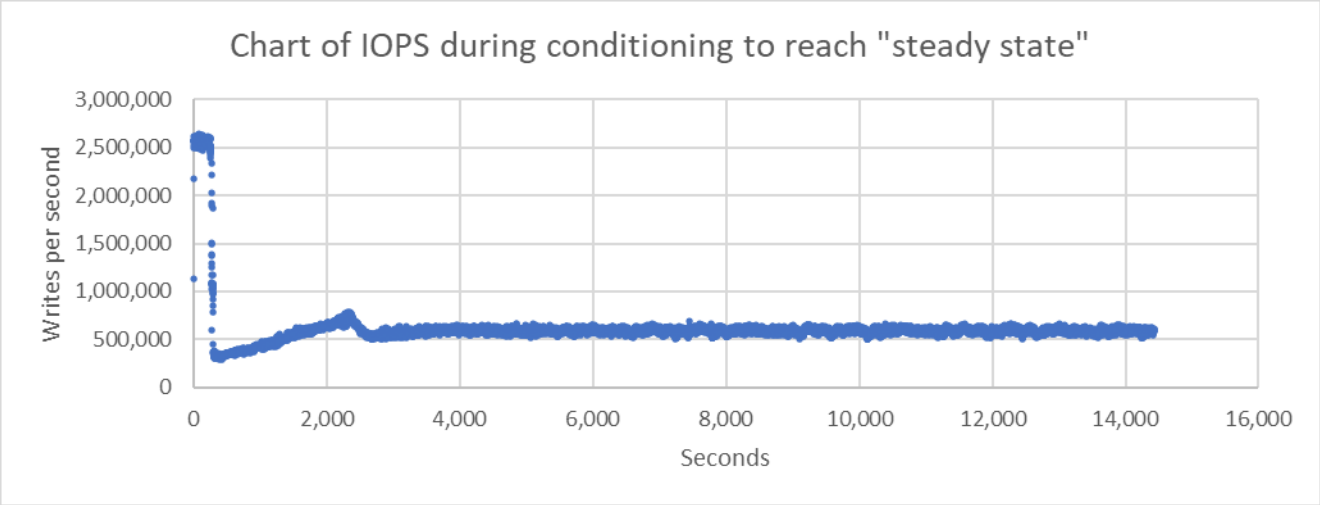
## 3.1 Conditioning the drive or volume before testing

Conditioning is important prior to running the performance testing. The devices should start testing from a steady state to allow accurate performance measurements. Conditioning involves writing blocks to the entire drive multiple times. We suggest separating the sequential workload tests and random workload tests to get more repeatable results. **It is crucial to condition the drives/volume for at least 4 hours (for larger volumes, this time may need to be increased) with writes of each applicable block size (e.g., 128k or 4k) before running the sequential tests and random tests. This means if you change block sizes during the testing procedure, we recommend preconditioning again to the new block size.** Getting steady state or stable performance is critical to avoid latency spikes or other performance bottlenecks.

**Note:** Conditioning is not required on Intel® Optane™ SSDs.

The sample fio script included in this document is configured for pre-conditioning.

The sample chart below represents IOPS during conditioning. The IOPS start really high, then drop down quite a bit, then eventually stabilize reaching steady state. This is why conditioning is important before starting the actual performance testing. The data here is just an example. Results will vary depending on your setup.



## 3.2 Notes about FIO and some parameters

Adding CPU affinity may be helpful. For example, if the SSDs are on the 2nd CPU (CPU NUMA Node 1), and each processor contains 24 cores, then the command to kick off fio would be "`numactl --cpunodebind=1 fio ...`". This command would also contain fio parameters of "`--cpus_allowed_policy=split --cpus_allowed=24-47`". To view the CPU ranges for each CPU NUMA node, run "`lscpu | grep "NUMA node" | grep "CPU"`".

fio output explained:

https://tobert.github.io/post/2014-04-17-fio-output-explained.html

fio manual page:

https://linux.die.net/man/1/fio

... or type "`man fio > man_fio.txt`" in your system under test and copy the file to laptop/PC.

How fio works:

https://github.com/axboe/fio/blob/master/HOWTO

| FIO Parameter | Notes |
|---|---|
| `randrepeat=0` | The parameter "`randrepeat`" defaults to 1 (enabled) and should be disabled to get more accurate performance. If enabled, results can lead to false positives (e.g., IOps are much higher than the spec). |
| `random_generator=tausworthe64` | For large capacity SSDs like 4TB or 8TB, the default value of "`tausworthe`" for parameter "`random_generator`" could present errors. Forcing it to use "`tausworthe64`" will eliminate these errors. |
| `iodepth_batch=<iodepth>` | fio may limit the iodepth to 16 even if iodepth is set to a larger number. Adding this parameter insures all iodepths are utilized which increases IOPS and bandwidth. |
| `minimal` | This parameter outputs the results to semicolon-delimited values which can then be changed to comma-separated values by typing the following after testing has completed.<br>"`sed -i -e 's/;/,/g' <fio output file>`" |

## 3.3 Checking CPU Usage

Tools like `iostat` and `dstat` can be used to collect IOPS to compare with the results that fio outputs. These tools can also be used to collect overall system CPU usage. When using `iostat` or `dstat`, observe the CPU idle percentages. The outcome is an accurate value of CPU usage not active due to all of the supporting processes for RAID, drives, and other tools that may be used for background activities. These idle percentages can be subtracted from 100 to get actual CPU usage.

To install `iostat`, run "`yum install sysstat`". To install `dstat`, run "`yum install dstat`".

The sample fio script included in this document is configured to use `iostat`.

# 4 Sample FIO script

The following bash script will need to be saved as a .sh file. To be able to run the script, change the permissions by typing "`chmod 755 <script file>.sh`" or "`chmod 777 <script file>.sh`" to make it executable by the administrative user. You may have to run this script as root. **Note**: The values in the variables in this script can be changed based on your needs.

```
#!/bin/bash

# Adjust any of the variables below
numOfHoursToPrepDrive=4
# Variables fioRuntime and fioRamptime are in seconds
fioRuntime=600
fioRamptime=600
numOfIterations=3
# If unsure of what CPU the NVMe SSDs are on, set "affinitized=false"
affinitized=true
# If NVMe SSDs are installed on 1st CPU then set "numaCpu=0"
# If NVMe SSDs are installed on 2nd CPU then set "numaCpu=1"
numaCpu=1
# Adjust variable "fioDevice" to applicable device like /dev/nvme0n1 for passthrough
#     or /dev/md126 for VROC RAID volume
fioDevice=/dev/md126
if [[ "${fioDevice:0:7}" == "/dev/md" ]]; then
    # Variable "raidVolume" can be changed to whatever the RAID level is
    # (i.e. "RAID0" or "4DR0" or "RAID5" or "4DR5" or etc)
    raidVolume="RAID"
    # Set group_thread_cnt to 4 for RAID 5 volume if it exists
    if [ -f /sys/block/${fioDevice:5}/md/group_thread_cnt ]; then
        echo 4 > /sys/block/${fioDevice:5}/md/group_thread_cnt
    fi
else
    raidVolume="Passthrough"
fi

CreateFioConfigFile ()
{
    echo "[global]" > FioConfig.ini
    echo "time_based" >> FioConfig.ini
    echo "ioengine=libaio" >> FioConfig.ini
    echo "direct=1" >> FioConfig.ini
    echo "buffered=0" >> FioConfig.ini
    echo "norandommap" >> FioConfig.ini
    echo "refill_buffers" >> FioConfig.ini
    echo "stonewall" >> FioConfig.ini
    echo "disable_slat=1" >> FioConfig.ini
    echo "disable_lat=1" >> FioConfig.ini
    echo "disable_bw=1" >> FioConfig.ini
    echo "randrepeat=0" >> FioConfig.ini
    echo "thread" >> FioConfig.ini
    echo "unified_rw_reporting=0" >> FioConfig.ini
    echo "group_reporting" >> FioConfig.ini
    echo "do_verify=0" >> FioConfig.ini
    if [[ $affinitized == true ]]; then
        echo "cpus_allowed_policy=split" >> FioConfig.ini
        echo "cpus_allowed=${affinity}" >> FioConfig.ini
    fi
    if [[ "${SeqOrRandom}" == "Random" ]]; then
        echo "percentage_random=100" >> FioConfig.ini
        echo "random_generator=tausworthe64" >> FioConfig.ini
    fi
```

```
    echo "runtime=${tempRuntime}" >> FioConfig.ini
    echo "ramp_time=${tempRamptime}" >> FioConfig.ini
    echo "" >> FioConfig.ini
    echo "[${tempFioOutputStr}]" >> FioConfig.ini
    echo "size=100%" >> FioConfig.ini
    echo "filename=${fioDevice}" >> FioConfig.ini
    if [ $writePercent -eq 100 ] && [[ "${SeqOrRandom}" == "Seq" ]]; then
        echo "rw=write" >> FioConfig.ini
    elif [ $writePercent -eq 0 ] && [[ "${SeqOrRandom}" == "Seq" ]]; then
        echo "rw=read" >> FioConfig.ini
    elif [ $writePercent -eq 100 ] && [[ "${SeqOrRandom}" == "Random" ]]; then
        echo "rw=randwrite" >> FioConfig.ini
        echo "rwmixwrite=100" >> FioConfig.ini
    elif [ $writePercent -eq 0 ] && [[ "${SeqOrRandom}" == "Random" ]]; then
        echo "rw=randread" >> FioConfig.ini
        echo "rwmixread=100" >> FioConfig.ini
    else
        echo "rw=randrw" >> FioConfig.ini
        echo "rwmixwrite=${writePercent}" >> FioConfig.ini
    fi
    echo "bs=${blockSize}" >> FioConfig.ini
    echo "offset=0" >> FioConfig.ini
    echo "iodepth=${iodepth}" >> FioConfig.ini
    echo "iodepth_batch=${iodepth}" >> FioConfig.ini
    echo "numjobs=${workers}" >> FioConfig.ini
    sleep 1
}

RunFioTest ()
{
    iostatRuntime=$(( tempRuntime + tempRamptime - 2 ))
    tempFioOutputStr="${configName}_${blockSize}_${SeqOrRandom}_${writePercent}%Writes"
    tempFioOutputStr="${tempFioOutputStr}_IODepth${iodepth}_Workers${workers}_${iteration}"
    iostatFile="./${resultsFolder}/${tempFioOutputStr}_iostat.txt"
    CreateFioConfigFile
    cp -a ./FioConfig.ini ./${resultsFolder}/${tempFioOutputStr}.ini
    fioParams="--minimal --eta=never"
    fioParams="${fioParams} --output=./${resultsFolder}/${tempFioOutputStr}.csv"
    echo "$(date +%x_%T)__FioTest_${tempFioOutputStr}"
    iostat -tmyx -p ${fioDevice} 1 ${iostatRuntime} > ${iostatFile} &
    $numactlCmd fio ${fioParams} ./FioConfig.ini &
    wait $(pidof fio)
    # The commands below change the results from semicolon delimited to comma delimited
    sed -i -e 's/;/,/g' ./${resultsFolder}/${tempFioOutputStr}.csv
    sed -i -e 's/_/,/g' ./${resultsFolder}/${tempFioOutputStr}.csv
}


################################################################################

# Check if device is present.  If not, then exit script.
if [ ! -b $fioDevice ]; then
    echo "The device specified in the fioDevice variable is not present."
    echo "Please adjust this variable in this script and try again."
    exit 1
fi

# Stop irqbalance service (this may help reduce high standard deviations between iterations)
systemctl stop irqbalance

# Setting CPU affinity command and applicable CPUs if variable "affinitized=true"
numactlCmd=""
if [[ $affinitized == true ]]; then
```

```
    numactlCmd="numactl --cpunodebind=$numaCpu"
    numOfNumaNodes=$(( $( lscpu | grep "NUMA node(s):" | rev | awk '{print $1}' | rev ) ))
    numaNodesStrings=( $( lscpu | grep "NUMA node" | grep "CPU" | awk '{print $2}' ) )
    if [ $numaCpu -ge $numOfNumaNodes ] && [ $numOfNumaNodes -ge 1 ]; then
        numaCpu=$(( numOfNumaNodes - 1 ))
    fi
    if [ $numaCpu -lt $numOfNumaNodes ]; then
        affinity=$( lscpu | grep "NUMA ${numaNodesStrings[$numaCpu]}" | awk '{print $4}' )
    fi
fi

configName="VROC-${raidVolume}"
timestamp=$(date +%Y_%m%d_%H%M)
resultsFolder=${timestamp}_${configName}
mkdir -p ./${resultsFolder}

# Start of sequential workload testing section
SeqOrRandom=Seq
for blockSize in 128k; do
    workers=1
    # Start of pre-conditioning section
    iteration=Prep; writePercent=100; iodepth=128
    tempRuntime=$(( numOfHoursToPrepDrive * 60 * 60 )); tempRamptime=30
    RunFioTest
    # End of pre-conditioning section
    for (( iteration=1; iteration<=$numOfIterations; iteration++ )); do
        for writePercent in 100 0; do
            # Modify the iodepths below based on your needs
            for iodepth in 128; do
                tempRuntime=${fioRuntime}; tempRamptime=${fioRamptime}
                RunFioTest
            done
        done
    done
done
# End of sequential workload testing section

# Start of random workload testing section
SeqOrRandom=Random
for blockSize in 4k; do
    # Start of pre-conditioning section
    iteration=Prep; writePercent=100; workers=8; iodepth=32
    tempRuntime=$(( numOfHoursToPrepDrive * 60 * 60 )); tempRamptime=30
    RunFioTest
    # End of pre-conditioning section
    for (( iteration=1; iteration<=$numOfIterations; iteration++ )); do
        for writePercent in 100 30 0; do
            for workers in 16 8; do
                # Modify the iodepths below based on your needs
                if [ $workers -eq 16 ]; then
                    iodepths=(256 128 64 32 16 8)
                elif [ $workers -eq 8 ]; then
                    iodepths=(256 128 64 32 16 8)
                elif [ $workers -eq 4 ]; then
                    iodepths=(256 128 64)
                else
                    iodepths=(256 128 64 32 16 8)
                fi
                for iodepth in "${iodepths[@]}"; do
                    tempRuntime=${fioRuntime}; tempRamptime=${fioRamptime}
                    RunFioTest
                done
            done
```

```
        done
    done
done
# End of random workload testing section

# Create the header for the CSV file
sourceString="fio_version,jobname,groupid,"
replacementStr="fio_version,jobname,blocksize,SeqOrRandom,write_percent"
replacementStr="${replacementStr},iodepth,threads,iteration,groupid,"
fioHeaderArray=( $( man fio | grep "terse_version" | sed "s/;/,/g" | sed
   "s%${sourceString}%${replacementStr}%g" ) )
fioHeader="${fioHeaderArray[@]}"
# Combining all results into one file (excluding any files from pre-conditioning)
echo "${fioHeader}" > ./${resultsFolder}/AllFioResults.csv
cat ./${resultsFolder}/*IODepth*[0-9].csv >> ./${resultsFolder}/AllFioResults.csv
# Convert some of the values to integers for the CSV file
sourceString="%Writes"
sed -i -e "s/${sourceString}//g" ./${resultsFolder}/AllFioResults.csv
sourceString="IODepth"
sed -i -e "s/${sourceString}//g" ./${resultsFolder}/AllFioResults.csv
sourceString="Workers"
sed -i -e "s/${sourceString}//g" ./${resultsFolder}/AllFioResults.csv

# Start irqbalance service back up
systemctl start irqbalance
```

# 5 Analyzing the Results

With the recommendations from this document, performing at least three iterations of each preferred workload should produce repeatable results. Once the results are viewed from Microsoft* Excel* and all rows are sorted to the applicable workloads, columns can be added for determining averages and standard deviations. To determine the standard deviation percentage for a group of cells in Excel, take the standard deviation of that group of cells and divide it by the average of the same group of cells and convert to a percentage format. This standard deviation percentage should indicate within 5%. If it shows a high variance across multiple workloads, you may need to check drive firmware or health of each drive in your RAID volume. If the standard deviation percentage is high in the beginning of the testing (after the conditioning runs), then conditioning of the drive/volume may need to be extended.

After looking at the overall performance results, if they appear to be low, it could be that the device is not being fully saturated. Increase the `numjobs` (threads) to see if this increases the performance but keep in mind that this could impact latency. Looking at the `dmesg` logs may help determining any bottlenecks as well.

Below are details of the output of the fio results when the output format is set to terse ("`--minimal`" parameter). Source: fio manual page.

```
# If using the script above, the headers will look like the below
terse version, fio version, jobname, block size, Sequential/Random, write percent,
   iodepth, threads, iteration, groupid, error
# If not using the script above, the headers will look like the below
terse version, fio version, jobname, groupid, error

READ status:
  Total IO (KiB), bandwidth (KiB/sec), IOPS, runtime (msec)
  Submission latency: min, max, mean, stdev (usec)
  Completion latency: min, max, mean, stdev (usec)
  Completion latency percentiles: 20 fields (see below)
  Total latency: min, max, mean, stdev (usec)
  Bw (KiB/s): min, max, aggregate percentage of total, mean, stdev, number of samples
  IOPS: min, max, mean, stdev, number of samples

WRITE status:
  Total IO (KiB), bandwidth (KiB/sec), IOPS, runtime (msec)
  Submission latency: min, max, mean, stdev (usec)
  Completion latency: min, max, mean, stdev (usec)
  Completion latency percentiles: 20 fields (see below)
  Total latency: min, max, mean, stdev (usec)
  Bw (KiB/s): min, max, aggregate percentage of total, mean, stdev, number of samples
  IOPS: min, max, mean, stdev, number of samples

TRIM status:
  Fields are similar to READ/WRITE status.

CPU usage:
  user, system, context switches, major faults, minor faults

I/O depths:
  <=1, 2, 4, 8, 16, 32, >=64

I/O latencies microseconds:
  <=2, 4, 10, 20, 50, 100, 250, 500, 750, 1000

I/O latencies milliseconds:
  <=2, 4, 10, 20, 50, 100, 250, 500, 750, 1000, 2000, >=2000

Disk utilization:
  disk name, read ios, write ios, read merges, write merges, read ticks, write ticks,
   time spent in queue, disk utilization percentage
```

```
Completion latency percentiles can be a grouping of up to 20 sets, so for the terse
   output fio writes all of them. Each field will look like this:
  1.00%=6112
which is the Xth percentile, and the 'usec' latency associated with it.

For Disk utilization, all disks used by fio are shown. So for each disk there will be a
   disk utilization section.
```

## 5.1 Test Results (Intel® VROC using 4 x 3.84TB Intel® D7-P5510 PCIe 4.0 U.2 SSDs)

Results below include IOPS, Bandwidth, and overall CPU usage. **Note:** The following results are based on a Coyote Pass system.

System configuration based on testing performed on August 5, 2021:

Intel® Server Board M50CYP Family, BIOS Version: SE5C620.86B.01.01.0003.2104260124, BIOS Release Date: 04/26/2021
Intel® VROC Pre-OS Version: 7.5.0.1152
CPUs: 2 x Intel® Xeon® Platinum 8358 CPU @ 2.60GHz (32 cores each)
RAM: 128GB RAM (16 x 8GB Micron* DDR4 3200MT/s SDRAM RDIMMs)
OS: Red Hat* Enterprise Linux Server 8.1, Kernel: 4.18.0-147.el8.x86_64
mdadm version: mdadm - v4.1 - 2018-10-01
kmod-iavmd-1.0.0.1494-rhel_81.x86_64.rpm package installed
Added "initcall_blacklist=vmd_drv_init pci=pcie_bus_perf" to grub boot option which disables inbox VMD, loads iavmd, and maximizes MaxPayload for each NVMe device
4 x 3.84TB Intel® D7-P5510 Series PCIe 4.0 U.2 SSDs (Model: SSDPF2KX038TZ, Firmware: JCV10016)

| | VROC 4DR0 | VROC 4DR5 | VROC 2DR1 | VROC 4DR10 |
|---|---|---|---|---|
| Workload | Ave. IOps | Ave. IOps | Ave. IOps | Ave. IOps |
| 4k_Random_100%-Write_16-Threads_256-IOdepth | 585,964 | 212,889 | 147,473 | 292,694 |
| 4k_Random_30%-Write_16-Threads_256-IOdepth | 1,076,204 | 558,165 | 347,395 | 692,866 |
| 4k_Random_0%-Write_16-Threads_256-IOdepth | 2,811,619 | 2,809,537 | 1,407,170 | 2,813,474 |

| Workload | Ave. MB/sec | Ave. MB/sec | Ave. MB/sec | Ave. MB/sec |
|---|---|---|---|---|
| 4k_Random_100%-Write_16-Threads_256-IOdepth | 2,344 | 852 | 590 | 1,171 |
| 4k_Random_30%-Write_16-Threads_256-IOdepth | 4,305 | 2,233 | 1,390 | 2,771 |
| 4k_Random_0%-Write_16-Threads_256-IOdepth | 11,247 | 11,238 | 5,629 | 11,254 |

| Workload | CPU Usage | CPU Usage | CPU Usage | CPU Usage |
|---|---|---|---|---|
| 4k_Random_100%-Write_16-Threads_256-IOdepth | 2.71% | 6.43% | 11.91% | 12.06% |
| 4k_Random_30%-Write_16-Threads_256-IOdepth | 2.97% | 8.23% | 12.04% | 12.18% |
| 4k_Random_0%-Write_16-Threads_256-IOdepth | 5.97% | 7.35% | 4.18% | 8.12% |