Flash Performance Best Practices

17 April 2018

Contents

1 Introduction

This document describes the server settings, configurations, and other recommendations for optimal performance of a flash system.

1.1 Performance Considerations

These settings and configurations produced optimal results for this particular system and specific tests. Your system and applications may require different or additional tuning to produce the best results for your particular use case. Performance tuning is often a tradeoff; over-optimizing for one performance attribute may have negative effects on another. You should not try to achieve peak performance for all raw block I/O patterns. You should tune performance for your particular use case and application. Each attribute (for example read latency, write IOPS, or read bandwidth) has a specific test that is tuned to determine the peak performance for just that attribute. For example, the maximum random write IOPS are achieved by testing 4KB writes. However, to achieve the highest bandwidth possible, 1M blocks of data are used. For example, to achieve peak sequential write IOPS performance, an SSD device must be used by an application that is accessing small chunks (4KB) of data. While that specific configuration may achieve peak IOPS on a device, that I/O pattern may not achieve peak bandwidth on the device. Meanwhile, the same device may achieve peak bandwidth when it is used by a different application that is reading large chunks of data (such as video editing or streaming). You should test and tune your system for the performance attribute or attributes that provide the greatest impact for your use case. Otherwise you may limit application performance by focusing on improving attributes that don't impact your application.

Data Destructive Tests

Write tests (including pre-conditioning) will write data to the device. These tests will write over all the data on the device. Only perform write tests on new or unused devices.

2 General Performance Optimization Settings

Obtaining peak performance requires some system-level tuning. The following are performance tuning actions that we recommend.

2.1 CPU Settings

2.1.1 Disable CPU Frequency Scaling

Dynamic Voltage and Frequency Scaling (DVFS) are power management techniques that adjust the CPU voltage and/or frequency to reduce power consumption by the CPU. These techniques help conserve power and reduce the heat generated by the CPU, but they adversely affect performance while the CPU transitions between low-power and high performance states.

These power-savings techniques are known to have a negative impact on I/O latency and IOPS. When tuning for performance, you may benefit from reducing or disabling DVSF completely, even though this may increase power consumption.

DVFS, if available, is often configurable as part of your operating system's power management features as well as within your system's BIOS interface. Within the operating system and BIOS, DVFS features are often found under the Advanced Configuration and Power Interface (ACPI) sections; consult your computer documentation for details.

2.1.2 Set CPU Frequency to Highest Level

If your BIOS allows you to set a fixed CPU frequency, make sure the frequency always stays at the highest level. Again, this will ensure that the CPU always stays at the highest frequency, and no performance is lost due to the transitioning between frequencies.

2.1.3 Disable CPU Power Throttling

Newer processors have the ability to go into lower power modes when they are not fully utilized. These idle states are known as ACPI C-states. The C0 state is the normal, full power, operating state. Higher C-states (C1, C2, C3, etc.) are lower power states.

While ACPI C-states save on power, they can have a negative impact on I/O latency and maximum IOPS. With each higher C-state, typically more processor functions are limited to save power, and it takes time to restore the processor to the C0 state.

When tuning for maximum performance you may benefit from limiting the C-states or turning them off completely, even though this may increase power consumption.

2.2 ACPI Mode

Use the ACPI to have the OS tell the hardware not to enter power save mode. If the HW supports CPU frequency scaling and such, the CPU core may scale down the frequency to about half, and then scale back up based on workload demand. However, each state transition may require some nominal time to transition, which can impact application latencies. For high QD tests, little to no difference may be observed. For low QD tests, some difference may be observed.

Using the Add/Remove Software GUI interface, or the RPM manager of your choice, add the `cpupowerutils` RPM. Then issue `x86_energy_perf_policy -r` to check the current setting. If needed, issue `x86_energy_perf_policy performance` to set the mode to max performance. Reboot the machine if necessary to have the new settings take effect.

Setting ACPI C-State Options

If your processor has ACPI C-states available, you can typically limit or disable them in the BIOS interface (sometimes referred to as a Setup Utility). ACPI C-states may be part of the Advanced Configuration and Power Interface (ACPI) menu. Consult your computer documentation for details.

C-States Under Linux

Newer Linux kernels have drivers that may attempt to enable ACPI C-states even if they are disabled in the BIOS. You can limit the C-state in Linux (with or without the BIOS setting) by adding the following to the kernel boot options

```
intel_idle.max_cstate=0 processor.max_cstate=0 idle=poll
```

In this example, C-states are completely disabled and the processor is run at full throttle even when idle.

2.3 ASPM Mode

The PCIe link between the RC and DUT may also enter a power saving mode within a few microseconds of not being used. The power saving mode is L0s, and the full power mode is L0. Again, the transition time from L0s to L0 can require notable time, causing impacts to any performance measurements. By using the OS, the power saving mode can be disabled.

Check the current ASPM setting by issuing `cat /sys/module/pcie_aspm/parameters/policy` and noting the output. A result of powersave means the L0s can be enabled, and a result of performance means the L0s is disable.

Edit the kernel parameters. Often, the kernel parameters may be located at /boot/grub/grub.conf. Find the applicable kernel entry, go to the kernel line, and add at the end:

```
pcie_aspm.policy=performance
```

Reboot the server and recheck the current ASPM setting to verify performance mode.

2.4 Fan Speeds

To prevent thermal throttling, set your server fan speeds to high. If your BIOS has a High Performance/Power Mode,enable it.

Power-saving Modes

You should also disable any power-saving modes. This will prevent the system from lowering the fan speeds. It will also prevent operating systems and the BIOS from suspending PCIe devices (using ASPM).

2.5 SELINUX Configuration

Sometimes the changes to the OS conflict with the security policy or cause additional overhead. It is suggested to turn off the security policy and rely on the security policy of the network, or if necessary, disable the network connectivity.

1. The SELINUX configuration policy is located at `/etc/selinux/config`.
2. Set the policy to `disabled` and `targeted`

```
# This file controls the state of SELinux on the system.
#SELINUX= can take one of these three values:
# enforcing - SELinux security policy is enforced.
# permissive - SELinux prints warnings instead of enforcing.
# disabled - No SELinux policy is loaded.
#SELINUX=enforcing
SELINUX=disabled
# SELINUXTYPE= can take one of these two values:
```

```
# targeted - Targeted processes are protected,

# mls - Multi Level Security protection.

SELINUXTYPE=targeted
```

2.6 Use Asynchronous I/O

To improve throughput, latency, and responsiveness, use asynchronous input/output (sometimes called AIO, async I/O, or non-blocking I/O) with your application. This advanced approach to input and output operations will allow your operating system to complete other operations while it waits for an I/O operation to complete.

Without this enabled, the system resources will remain idle until each requested I/O operation is completed. This traditional approach is called synchronous I/O or blocking I/O.

2.7 Use Direct I/O

Direct I/O bypasses the OS page cache and allows applications to read and write directly from devices.

About the Page Cache

Traditional I/O paths include the page cache. The page cache is a DRAM cache of buffered data from storage. By buffering data (both reads and writes) from storage devices in memory, the system can often avoid the time-consuming disk seeks and slower speeds of those legacy disk drives. Typically, all memory that is not allocated to applications is used for the page cache.

Page caching is why increasing RAM on a system usually increases the speed and responsiveness of I/O on that system. With a larger caching space, more I/Os can be buffered by the RAM. SSD devices are fast enough that this buffering in DRAM is often detrimental to performance.

About Direct I/O

Using direct I/O to bypass the page cache provides the following benefits:

- Less complex I/O path
- Lower overall CPU utilization
- Less host memory capacity and bandwidth usage

In most cases, direct I/O increases performance for SSD based systems, but you should benchmark your application to verify this. Some workloads that don't use AIO, threads, or multiple processes to create multiple outstanding requests may benefit from the page cache instead.

Many I/O-intensive applications have tunable parameters that control how they interact with the low-level I/O subsystem, including turning on direct I/O. Application tuning is outside of the scope of this document.

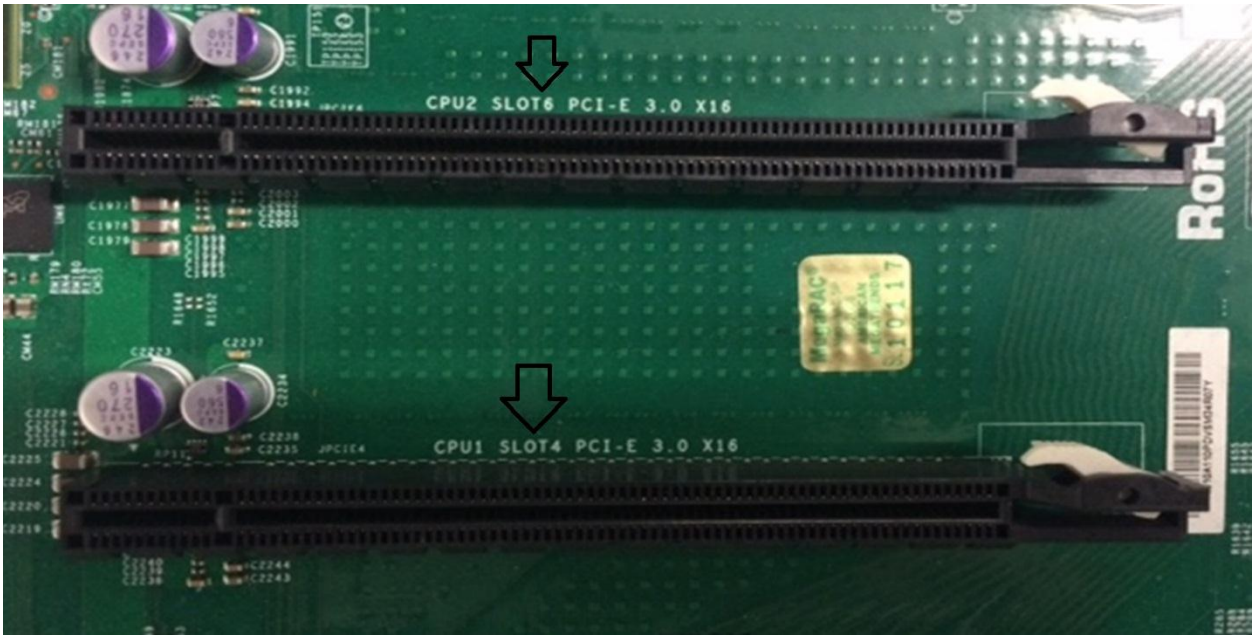
2.8 NUMA Node

If a multiple-socket CPU server is used, examine the motherboard and examine the PCIe slots. Many motherboards have silkscreen labels that indicate which PCIe slot is paired to the corresponding CPU socket. Servers with a NUMA (Non-Uniform Memory Access) architecture may require special installation instructions in order to maximize device performance. This includes most multi-socket servers.

On some servers with NUMA architecture, during system boot, the BIOS will not associate PCIe slots with the correct NUMA node. Incorrect mappings result in inefficient I/O handling that can significantly degrade performance.

The arrows indicate the CPU socket of the PCIe slot for the DUT.

Please refer to Determining the NUMA Node if the motherboard is not easily accessible or CPU node numbers are absent near the slots.



2.8.1 Setting NUMA Affinity

The performance can be maximized when the test runs on the CPU socket that is directly attached to the

PCIe slot of the DUT; however, if the performance test runs on a different CPU socket, then the communication must be switched between the sockets by using QPI processor-to-processor communication, and then to the PCIe slot of the DUT, which may cause additional latency and performance degradation.

If multiple CPU sockets are present on the server, determine the NUMA node that is associated with the

PCIe slot of the DUT.

1. Use the `lspci` command to find the DUT.
2. Note the `bus:device` numbers for the DUT.
3. Use `cat /sys/class/pci_bus/<bus:device>/device/numa_node` and note the output to determine the NUMA node.

2.8.2 Interrupt Handler Affinity

Device latency can be affected by placement of interrupts on NUMA systems. We recommend placing interrupts for a given device on the same NUMA node that the application is issuing I/O from. If the CPUs on this node are overwhelmed with user application tasks, in some cases it may benefit performance to move the interrupts to a remote node to help load-balance the system.

Many operating systems will attempt to dynamically place interrupts across the nodes, and generally make good decisions.

Linux IRQ Balancing

In Linux this dynamic placement is called IRQ Balancing. You can check to see if the IRQ balancer is effective by checking /proc/interrupts. If the interrupts are unbalanced (too many device interrupts on one node) or on an overwhelmed node, you may need to stop the IRQ balancer and manually distribute the interrupts in order to balance the load and improve performance.

Starting the IRQ Balancer may resolve interrupt affinity issues. For example, run the following command:

```
service irqbalance start
```

If that does not resolve the affinity issues, then we recommend manually pinning the device interrupts to specific nodes.

Hand-tuning interrupt placement in Linux is an advanced option that requires profiling of application performance on any given hardware. Please see your operating system documentation for information on how to pin specific device interrupts to specific nodes.

2.9 MTU Setting

For best performance, set the Maximum Transmissions Unit (MTU) network setting at 5000 or higher (Jumbo size is 9216). Set the same MTU value on the host, the switch and the storage unit, for compatibility.

3 System Configuration

This section describes an example system configuration that may be used to produce performance data.

3.1 Hardware

The performance differences between a system or server can be notable. For the purposes of this document, we will assume a traditional enterprise server for simplicity.

It is recommended to select a high end enterprise server for performance measurements. For simplicity, a single socket CPU (with multiple CPU cores) will make things easy, although a multiple socket CPU server will also suffice. A Haswell CPU is recommended for best performance, along with a reasonably high CPU frequency (2.2GHz+), sufficient DRAM (16GB+) and DRAM frequency.

When connecting a PCIe SSD to a server's motherboard, double check that the server's PCIe slot's electrical width provides sufficient bandwidth for the DUT. For example, some PCIe slots provide a x8 mechanical connector, but only a x4 electrical connector. While a PCIe x8 device can physically plug into the motherboard's slot, it will only operate in x4 mode. The reason that server's slot may not be fully electrically connected is that the total PCIe bandwidth of the CPU or North Bridge may be limited, and also that the routing of PCIe signal lines on the PCB is challenging for space and signal integrity reasons.

Here is a sample configuration:

Component	Quantity	Description
Server	1	Dell PowerEdge R730xd
CPU	88	Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz
RAM	125GB	DDR4 2400 MHz
PCIe Slots	3	

HDD	1	2TB SATA Hard Drive

3.2 Software

Operating System: CentOS

I/O Tool: fio version 3.5 or higher

3.2.1 Installing fio

We recommend testing the performance of your device(s) using the fio utility. The fio utility is an open source application; it is not created or maintained by HGST. It stands for "Flexible I/O" Tester and is available at

http://freecode.com/projects/fio

Sample Installation

The fio utility is in active development, we recommend installing the latest version. These commands will make and install fio:

```
$ git clone git://git.kernel.dk/fio.git
$ cd fio
$ make
$ make install
$ cd
```

4 Testing

The fio I/O tool uses .ini configuration files to define specific I/O benchmark patterns. Command-line parameters can also be used, however .ini configuration files reduce the potential for user input error.

See Sample Linux fio Job Files in Appendix A for the test scripts to reproduce the performance metrics reported in the data sheets.

> **Sample fio Job Files in Appendix A**
>
> These are only sample test jobs. You may modify test jobs to test your use case and workloads. For example, if the I/O size for your workloads is 8K, and you are using a fio job with a block size of 4K, then you can re-configure the job for a 8K block size.

Please observe the following guidelines when performing raw I/O benchmarks:

- If run all together, these tests should be run in numerical order.
- Prior to each write test, detach the device, format it, and attach it.
- Prior to all read tests, the device needs to be pre-conditioned (i.e. filled with data). This is the fio-job-04 command/file. Once the device has been pre-conditioned, you may perform any number of read tests on the device.

> **Data Destructive Tests**
>
> Write tests (including pre-conditioning) will write data to the device. These tests will write over all the data on the device. Only perform write tests on new or unused devices.

- For the peak IOPS tests that are smaller than 4K, the device should have a formatted sector size of **512B**.
- For tests using block sizes greater than or equal to 4KB, format the device with a sector size of 4KB.
- Run each test script with the following command:

```
fio --output=fio.output.<number> fio-job-<number>.ini
```

- You may wish to enter the jobs directly as commands rather than creating and using .ini files. The direct commands are listed in the next sections.

Using .ini files

We recommend using the .ini files, as it will prevent input error. However, if you do enter the long commands directly, be careful when copying and pasting (especially from a PDF). **We recommend copying the direct commands into a simple text editor to make sure there are no additional line breaks.**

Example output:

```
/dev/vs_fio_test: (g=0): rw=rw, bs=512-512/512-512, ioengine=libaio,
iodepth=32
...
/dev/vs_fio_test: (g=0): rw=rw, bs=512-512/512-512, ioengine=libaio,
iodepth=32
fio-2.0.8
Starting 4 processes


/dev/vs_fio_test: (groupid=0, jobs=4): err= 0: pid=30918

Description : [fio sequential 512 write peak IOPS]

write: io=7018.6MB, bw=239540KB/s, iops=479075, runt= 30001msec

  slat (usec): min=15 , max=523 , avg=54.63, stdev=41.91
  clat (usec): min=1 , max=2059 , avg=206.50, stdev=78.85
   lat (usec): min=35 , max=2142 , avg=261.18, stdev=84.16
  clat percentiles (usec):
    | 1.00th=[ 70], 5.00th=[ 102], 10.00th=[ 120], 20.00th=[ 143],
    | 30.00th=[ 161], 40.00th=[ 179], 50.00th=[ 195], 60.00th=[ 215],
    | 70.00th=[ 237], 80.00th=[ 266], 90.00th=[ 306], 95.00th=[ 346],
    | 99.00th=[ 434], 99.50th=[ 474], 99.90th=[ 668], 99.95th=[ 820],

    | 99.99th=[ 1144]
  bw (KB/s) : min= 1, max=63280, per=24.58%, avg=58889.02, stdev=8769.50
  lat (usec) : 2=0.01%, 4=0.01%, 10=0.01%, 20=0.01%, 50=0.16%
  lat (usec) : 100=4.32%, 250=70.62%, 500=24.53%, 750=0.29%, 1000=0.05%
  lat (msec) : 2=0.02%, 4=0.01%
```

```
cpu : usr=7.20%, sys=45.61%, ctx=5305818, majf=0, minf=105
IO depths : 1=0.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.1%, 32=115.3%,
>=64=0.0%
    submit : 0=0.0%, 4=0.0%, 8=0.0%, 16=100.0%, 32=0.0%, 64=0.0%,
>=64=0.0%
   complete : 0=0.0%, 4=0.0%, 8=0.0%, 16=100.0%, 32=0.0%, 64=0.0%,
>=64=0.0%
   issued : total=r=0/w=14372752/d=0, short=r=0/w=0/d=0


Run status group 0 (all jobs):
  WRITE: io=7018.6MB, aggrb=239539KB/s, minb=239539KB/s,
maxb=239539KB/s, mint=30001msec, maxt=30001msec


Disk stats (read/write):
  vs_fio_test: ios=0/8637747, merge=0/0, ticks=0/1407990,
in_queue=1412150, util=100.00%
```

You can convert the bandwidth in this example to MB/s by dividing by 1024.

4.1 Sample Linux fio Job Commands

These Linux test commands are optimized for the sample test system (as described in **System Configuration** above). Optimal settings to test your system may vary depending on your system and target applications. For example, you may need to use a different `ioengine`; see the `fio` utility documentation for more information.

Testing Device

For every test, replace **/dev/vs_fio_test** with the particular device you wish to test.

Data Destructive Tests

Write tests (including pre-conditioning) will write data to the device. These tests will write over all the data on the device. Only perform write tests on new or unused devices.

Remove Line Breaks

These commands must be entered as a single line. Due to PDF formatting constraints, the jobs below contain forced line breaks. We recommend copying the direct commands into a simple text editor to make sure there are no line breaks. If the command extends beyond a page break, make sure you remove any footer information that you may copy over (document title and page number).

Sudo

By default the regular linux users cannot read or write to the block devices. You can use **sudo** command to override this security restriction and run fio on these devices as superuser:

$ sudo fio ...

4.1.1 fio-job-01 - Random 4KB Write Test for Latency

```
fio --readwrite=randrw --rwmixread=0 --blocksize=4k --ioengine=sync --
numjobs=1 --thread=0 --direct=1 --iodepth=1 --iodepth_batch=1 --
iodepth_batch_complete=1 --group_reporting=1 --ramp_time=5 --
norandommap=1 --description="fio random 4k write LATENCY" --time_based=1
--runtime=30 --randrepeat=0 --name=/dev/vs_fio_test  --cpus_allowed=1-4
--filename=/dev/vs_fio_test
```

4.1.2 fio-job-02 - Random 4K Write Test for Peak IOPS

```
fio --readwrite=randrw --rwmixread=0 --blocksize=4k --ioengine=libaio --
numjobs=4 --thread=0 --direct=1 --iodepth=128 --iodepth_batch=4 --
iodepth_batch_complete=4 --group_reporting=1 --ramp_time=5 --
norandommap=1 --description="fio random 4k write peak IOPS" --
time_based=1 --runtime=30 --randrepeat=0 --name=/dev/vs_fio_test --
cpus_allowed=1-4 --filename=/dev/vs_fio_test
```

4.1.3 fio-job-03 - Random 1M Write Test for Peak Bandwidth

```
fio --readwrite=randrw --rwmixread=0 --blocksize=1M --ioengine=libaio --
numjobs=4 --thread=0 --direct=1 --iodepth=128 --iodepth_batch=4 --
iodepth_batch_complete=4 --group_reporting=1 --ramp_time=5 --
norandommap=1 --description="fio random 1M write peak BW" --time_based=1
--runtime=30 --randrepeat=0 --name=/dev/vs_fio_test --cpus_allowed=1-4 -
-filename=/dev/vs_fio_test
```

4.1.4 fio-job-04 - (PRECONDITION) Sequential 1M Complete Write Test

```
fio --readwrite=write --rwmixread=0 --blocksize=1M --ioengine=libaio --
thread=0 --size=100% --iodepth=16 --group_reporting=1 --description="fio
PRECONDITION sequential 1M complete write" --name=/dev/vs_fio_test --
cpus_allowed=1-4 --filename=/dev/vs_fio_test
```

4.1.5 fio-job-05 - Random 4KB Read Test for Latency

```
fio --readwrite=randrw --rwmixread=100 --blocksize=4k --ioengine=sync --
numjobs=1 --thread=0 --direct=1 --iodepth=1 --iodepth_batch=1 --
iodepth_batch_complete=1 --group_reporting=1 --ramp_time=5 --
norandommap=1 --description="fio random 4k read LATENCY" --time_based=1
--runtime=30 --randrepeat=0 --name=/dev/vs_fio_test --cpus_allowed=1-4 -
-filename=/dev/vs_fio_test
```

4.1.6 fio-job-06 - Random 4K Read Test for Peak IOPS

```
fio --readwrite=randrw --rwmixread=100 --blocksize=4k --ioengine=libaio
--numjobs=4 --thread=0 --direct=1 --iodepth=128 --iodepth_batch=4 --
iodepth_batch_complete=4 --group_reporting=1 --ramp_time=5 --
norandommap=1 --description="fio random 4k read peak IOPS" --
```

```
time_based=1 --runtime=30 --randrepeat=0 --name=/dev/vs_fio_test --
cpus_allowed=1-4 --filename=/dev/vs_fio_test
```

4.1.7 fio-job-07 - Random 1M Read Test for Peak Bandwidth

```
fio --readwrite=randrw --rwmixread=100 --blocksize=1M --ioengine=libaio
--numjobs=4 --thread=0 --direct=1 --iodepth=128 --iodepth_batch=4 --
iodepth_batch_complete=4 --group_reporting=1 --ramp_time=5 --
norandommap=1 --description="fio random 1M read peak BW" --time_based=1
--runtime=30 --randrepeat=0 --name=/dev/vs_fio_test  --cpus_allowed=1-4
--filename=/dev/vs_fio_test
```

5 Appendix A - Sample Linux fio Job Files

These Linux test files are optimized for the sample test system (as described in **System Configuration** above). Optimal settings to test your system may vary depending on your system and target applications. For example, you may need to use a different `ioengine`; see the `fio` utility documentation for more information.

> **Data Destructive Tests**
>
> Write tests (including pre-conditioning) will write data to the device. These tests will write over all the data on the device. Only perform write tests on new or unused devices.

5.1.1 fio-job-01 - Random 4KB Write Test for Latency

To use this script:

1. Create a file with the script contents
2. Name it `fio-job-01.ini`
3. Replace `/dev/vs_fio_test` with the particular device you wish to test.
4. Follow the instructions in **Testing** chapter above

```
[global]
readwrite=randrw
rwmixread=0
blocksize=4k
ioengine=sync
numjobs=1
thread=0
direct=1
iodepth=1
iodepth_batch=1
iodepth_batch_complete=1
group_reporting=1
ramp_time=5
```

```
norandommap=1
description=fio random 4k write LATENCY
time_based=1
runtime=30
randrepeat=0

[/dev/vs_fio_test]
filename=/dev/vs_fio_test
cpus_allowed=1-4
```

5.1.2 fio-job-02 - Random 4K Write Test for Peak IOPS

To use this script:

1. Create a file with the script contents
2. Name it `fio-job-02.ini`
3. Replace `/dev/vs_fio_test` with the particular device you wish to test.
4. Follow the instructions in **Testing** chapter above

```
[global]
readwrite=randrw
rwmixread=0
blocksize=4k
ioengine=libaio
numjobs=4
thread=0
direct=1
iodepth=128
iodepth_batch=4
iodepth_batch_complete=4
group_reporting=1
ramp_time=5
```

```
norandommap=1
description=fio random 4k write peak IOPS
time_based=1
runtime=30
randrepeat=0

[/dev/vs_fio_test]
filename=/dev/vs_fio_test
cpus_allowed=1-4
```

5.1.3 fio-job-03 - Random 1M Write Test for Peak Bandwidth

To use this script:

1. Create a file with the script contents
2. Name it `fio-job-03.ini`
3. Replace `/dev/vs_fio_test` with the particular device you wish to test.
4. Follow the instructions in **Testing** chapter above

```
[global]
readwrite=randrw
rwmixread=0
blocksize=1M
ioengine=libaio
numjobs=4
thread=0
direct=1
iodepth=128
iodepth_batch=4
iodepth_batch_complete=4
group_reporting=1
ramp_time=5
```

```
norandommap=1
description=fio random 1M write peak BW
time_based=1
runtime=30
randrepeat=0

[/dev/vs_fio_test]
filename=/dev/vs_fio_test
cpus_allowed=1-4
```

5.1.4 fio-job-04 - (PRECONDITION) Sequential 1M Complete Write Test

To use this script:

1. Create a file with the script contents
2. Name it `fio-job-04.ini`
3. Replace `/dev/vs_fio_test` with the particular device you wish to test.
4. Follow the instructions in **Testing** chapter above

```
[global]
readwrite=write
rwmixread=0
blocksize=1M
ioengine=libaio
thread=0
size=100%
iodepth=16
group_reporting=1
description=fio PRECONDITION sequential 1M complete write

[/dev/vs_fio_test]
filename=/dev/vs_fio_test
```

```
cpus_allowed=1-4
```

5.1.5 fio-job-05 - Random 4KB Read Test for Latency

To use this script:

1. Create a file with the script contents
2. Name it `fio-job-05.ini`
3. Replace `/dev/vs_fio_test` with the particular device you wish to test.
4. Follow the instructions in **Testing** chapter above

```
[global]
readwrite=randrw
rwmixread=100
blocksize=4k
ioengine=sync
numjobs=1
thread=0
direct=1
iodepth=1
iodepth_batch=1
iodepth_batch_complete=1
group_reporting=1
ramp_time=5
norandommap=1
description=fio random 4k read LATENCY
time_based=1
runtime=30
randrepeat=0

[/dev/vs_fio_test]
filename=/dev/vs_fio_test
```

```
cpus_allowed=1-4
```

5.1.6 fio-job-06 - Random 4K Read Test for Peak IOPS

To use this script:

1. Create a file with the script contents
2. Name it `fio-job-06.ini`
3. Replace `/dev/vs_fio_test` with the particular device you wish to test.
4. Follow the instructions in **Testing** chapter above

```
[global]
readwrite=randrw
rwmixread=100
blocksize=4k
ioengine=libaio
numjobs=4
thread=0
direct=1
iodepth=128
iodepth_batch=4
iodepth_batch_complete=4
group_reporting=1
ramp_time=5
norandommap=1
description=fio random 4k read peak IOPS
time_based=1
runtime=30
randrepeat=0

[/dev/vs_fio_test]
filename=/dev/vs_fio_test
```

```
cpus_allowed=1-4
```

5.1.7 fio-job-07 - Random 1M Read Test for Peak Bandwidth

To use this script:

1. Create a file with the script contents
2. Name it `fio-job-07.ini`
3. Replace `/dev/vs_fio_test` with the particular device you wish to test.
4. Follow the instructions in **Testing** chapter above

```
[global]
readwrite=randrw
rwmixread=100
blocksize=1M
ioengine=libaio
numjobs=4
thread=0
direct=1
iodepth=128
iodepth_batch=4
iodepth_batch_complete=4
group_reporting=1
ramp_time=5
norandommap=1
description=fio random 1M read peak BW
time_based=1
runtime=30
randrepeat=0


[/dev/vs_fio_test]
filename=/dev/vs_fio_test
```

```
cpus_allowed=1-4
```

6 Appendix B - Testing and Tuning Strategies

Previous sections in this guide describe the system configurations and tests that may be used to produce peak performance data. You may wish to apply the principles of this guide to test a device on your particular system. While no synthetic workload can mimic the behavior of real applications, the considerations in this appendix will help you design tests that more accurately represent real-world use of a device.

Benchmarking Devices

SSD devices have distinct design differences from traditional "spinning" media. These differences result in reduced latency and increased bandwidth. Because devices perform differently from traditional devices, some of the traditional benchmarking tests and methodology may no longer be appropriate.

For example, some traditional tests are designed to incorporate disk seek time, which no longer applies to SSD devices. Many of the strategies in this appendix are created for the unique characteristics of SSD devices.

Precondition the Device

When a read command is issued to a device for a Logical Block Address (LBA) region that has never been written to, the device will automatically return all zeros. It will do this without accessing the NAND flash because it contains no data. This will result in speeds that are greater than the real-world capabilities of the device. To prevent this, be sure to "precondition" the device by writing to all of the LBAs that you intend to read from. For example, run fio-job-04. This will fill the device with random data and it will result in read tests that more closely resemble real-world use.

Writing Tests on a Full Device

Some benchmarking tests perform constant writes randomly across all regions of a device. This legacy strategy was designed to show how the seek time of the device impacts the performance of writes to random areas of the device. Because SSD devices have no variable seek latency, this test method is often unnecessary. It may not represent realistic use of the device unless your application uses a high percentage of "hot" data.

In real-world applications, the data on a storage device is often characterized by a mix of "hot" (changing) data and "cold" (relatively static) data. Completely filling the device and then running random write tests across the device simulates a scenario in which 90% or more of the data on the device is hot (continually being rewritten). This less common scenario will result in reduced performance unless the device is tuned for the heavy/constant write workload by increasing the reserve space on the device.

As writes occur and the device becomes full, the device relies on a reserved space (in addition to the advertised capacity) to perform maintenance on the data. This reserve space is large enough for most applications and uses but, as the percentage of hot data on a full device increases, the device must rely on the reserve space to wear level the device.

If a heavy write workload is important to your application, we recommend the following testing strategy:

1. Fill the device only until it reaches the total capacity that the application will use.
2. Determine your application's mix of hot and cold data.
3. Limit the benchmarking application to continually rewrite only the hot data portion.

Disk Aggregation (RAID) Considerations

Depending on your application, you may see a small, but measurable, I/O overhead that impacts IOPS when using Linux MD RAID, or other volume management techniques.

You can test for this overhead by benchmarking a single, aggregated RAID 0 (consisting of two or more devices) and comparing that output to the same benchmark settings run on just the devices (without the RAID configuration).

If application-layer striping or aggregation is available, you should benchmark both the application-layer aggregation and the Operating System layer aggregation, and then determine which works better for that application.

7 Appendix C - Debugging Performance Issues

CPU Thermal Throttling or Auto-Idling

Problem

Current-generation CPUs monitor themselves for thermal overheat. If the CPU is operating near its manufacturer's specified thermal tolerance, the CPU may reduce its clock rate in order to keep from overheating. This reduces overall system performance, including device performance.

Some systems have a CPU idle detector installed as well. This decreases the clock rate of the CPU under low load in order to conserve power, but it also decreases overall system performance, including device performance.

Solution

To detect a CPU running at a slower rate, inspect the contents of /proc/cpuinfo (in the Linux kernel) for discrepancies between the speed reported by the model and the current running speed. This must be done for each of the CPUs included in the listing.

Use the following command:

```
cat /proc/cpuinfo
```

Below is sample output of a system that has been throttled from 2.66 GHz to 1 GHz.

CPU throttling:

```
processor  : 0
vendor_id  : GenuineIntel
cpu family : 6
```

```
model      : 15
model name : Intel (R) Xeon (R) CPU 5150 @2.66GHz
stepping   : 5
cpu MHz    : 1000.382
cache size : 4096 KB
.
.
```