# Flexible I/O

## How fio works

Often, you might face a conundrum where you will need to comprehend and analyze how your system would perform with varying I/O loads and characteristics (i.e. how does your system perform when there are synchronous and asynchronous loads, what happens when an I/O buffer is involved, how the performance varies with I/O load coming from different sources, etc.) without having access to a platform that would enable you to scrutinize the same. This is where simulating and stress testing your system with I/O loads would come into the picture. FIO provides exactly this; by simulating I/O loads that your system will need to handle hassle-free not only in the real world but also real-time, you can verify as well as tune your system configuration to achieve optimal performance.

The first step in getting fio to simulate a desired I/O workload, is writing a job file describing that specific setup. A job file may contain any number of threads and/or files – the typical contents of the job file is a global section defining shared parameters, and one or more job sections describing the jobs involved. When run, fio parses this file and sets everything up as described. If we break down a job from top to bottom, it contains the following basic parameters:

## [I/O type](https://fio.readthedocs.io/en/latest/fio_doc.html#i-o-type)

Defines the I/O pattern issued to the file(s). Examples include Sequential Read/Write, Random Read/Write, Mixed, buffered I/O, direct/raw I/O.

## [Block size](https://fio.readthedocs.io/en/latest/fio_doc.html#block-size)

I/O chunk size (single or range of block sizes).

## [I/O size](https://fio.readthedocs.io/en/latest/fio_doc.html#i-o-size)

Amount of data read/written.

## [I/O engine](https://fio.readthedocs.io/en/latest/fio_doc.html#i-o-engine)

I/O source. Could be through memory mapping the file, using regular read/write, splice, async I/O, or even SG (SCSI generic sg).

## [I/O depth](https://fio.readthedocs.io/en/latest/fio_doc.html#i-o-depth)

Queuing depth for aysnc I/O engine.

## [Target file/device](https://fio.readthedocs.io/en/latest/fio_doc.html#target-file-device)

Number of files/devices workload is being targeted to.

## [Threads, processes and job synchronization](https://fio.readthedocs.io/en/latest/fio_doc.html#threads-processes-and-job-synchronization)

Number of threads or processes workload should be spread over.

The above are the basic parameters defined for a workload, in addition there’s a multitude of parameters that modify other aspects of how this job behaves.

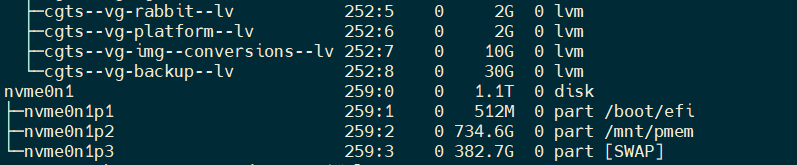
## Interpreting the output:

While executing fio files, the characters inside the first set of square brackets denote the status (current) of each thread. The first character is the first job defined in the job file, and so forth. The possible values (in typical life cycle order) are:

| **Idle** | **Run** |  |
| --- | --- | --- |
| P |  | Thread setup, but not started. |
| C |  | Thread created. |
| I |  | Thread initialized, waiting, or generating necessary data. |
|  | p | Thread running pre-reading file(s). |
|  | / | Thread is in ramp period. |
|  | R | Running, doing sequential reads. |
|  | r | Running, doing random reads. |
|  | W | Running, doing sequential writes. |
|  | w | Running, doing random writes. |
|  | M | Running, doing mixed sequential reads/writes. |
|  | m | Running, doing mixed random reads/writes. |
|  | D | Running, doing sequential trims. |
|  | d | Running, doing random trims. |
|  | F | Running, currently waiting for *fsync(2)*. |
|  | V | Running, doing verification of written data. |
| f |  | Thread finishing. |
| E |  | Thread exited, not reaped by main thread yet. |
| \_ |  | Thread reaped. |
| X |  | Thread reaped, exited with an error. |
| K |  | Thread reaped, exited due to signal. |

**File system test:**

Mounted device : /dev/ nvme0n1p2 /mnt/pmem

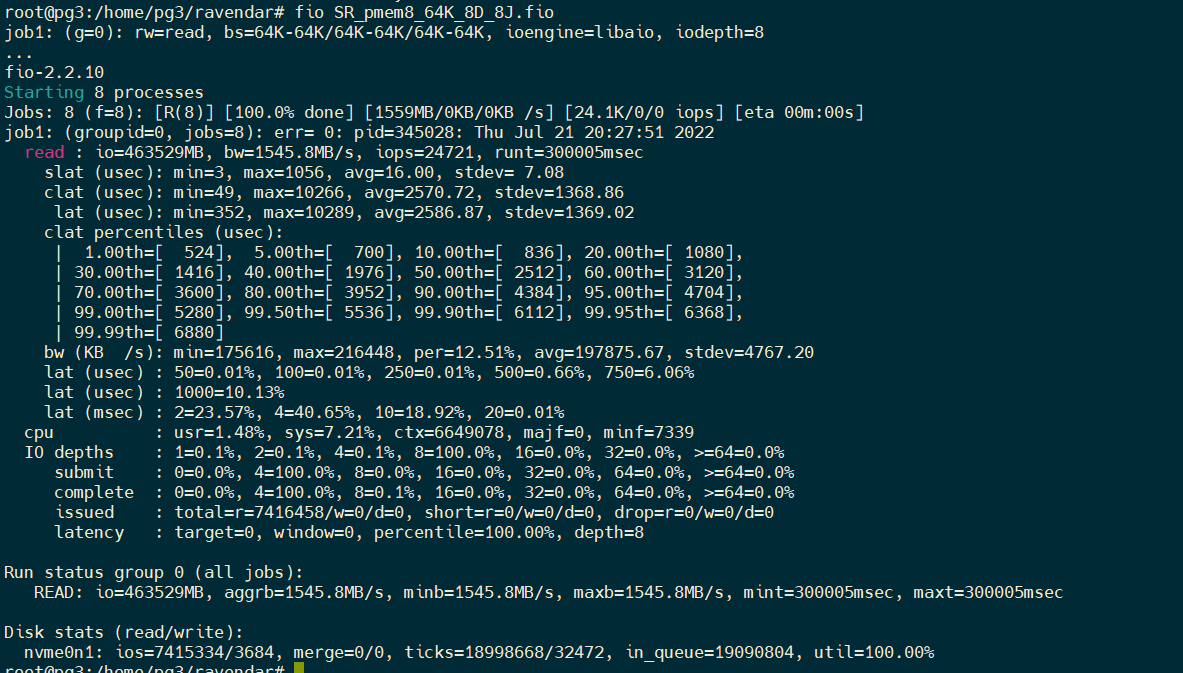


**Test Run for Sequential read :**

|  |
| --- |
| [global]  time\_based  group\_reporting  norandommap  ioengine=libaio  direct=1  buffered=0  size=10g  randrepeat=0  refill\_buffers  numjobs=8  rw=read  iodepth=8  bs=64k  runtime=300  [job1]  #directory=/mnt/pmem8  directory=/mnt/pmem |

**Output:**

When fio is done it will show the data for each thread, group of threads, and disks in that order. For each overall thread (or group) the output looks like:



**read** string gives the statistical analysis of

**I/O** is total performed inputs and outputs in this case its 463529MB

( in power of 2 format / **runtime** of that thread).

**IOPS** is the average I/Os performed per second here iops is 24721.

**BW** is the average bandwidth rate shown as:

{value in power of 2 format (value in power of 10 format). }

Here the bandwidth is 1545.8MB/s

**Slat**

submission latency is the time it took to submit the I/O. Fio will choose the most appropriate base and print that (in the example above nanoseconds was the best scale).

**clat**

Completion latency denotes the time from submission to completion of the I/O pieces

**lat**

Total latency denotes the time from when fio created the I/O unit to completion of the I/O operation.

**bw**

Bandwidth statistics based on samples. Same names as the slat stats, but also includes the number of samples taken (**samples**) and an approximate percentage of total aggregate bandwidth this thread received in its group (**per**). This last value is only really useful if the threads in this group are on the same disk, since they are then competing for disk access.

**lat (nsec/usec/msec)**

The distribution of I/O completion latencies from initiation to completion. Unlike the separate read/write sections above, the data here and in the remaining sections apply to all I/Os for the reporting group.

50=0.01% means that 0.01% of the I/Os completed in under 50us. 100=0.01% means that .01% of the I/Os required 50 to 99us for completion and so on.

**cpu**

CPU usage of User(usr) and system time(sys), along with the number of context switches this thread went through(ctx), and finally the number of major(mj f) and minor page faults(minf). The CPU utilization numbers are averages for the jobs in that reporting group, while the context and fault counters are summed.

**IO depths**

The distribution of I/O depths over the job lifetime. The numbers are divided into powers of 2 and each entry covers depths from that value up to those that are lower than the next entry

e.g., 16= covers depths from 16 to 31. Note that the range covered by a depth distribution entry can be different to the range covered by the equivalent submit/complete distribution entry.

**IO submit**

How many pieces of I/O were submitting in a single submit call. Each entry denotes that amount and below, until the previous entry

e.g., 4=100% means that we submitted anywhere between 0 to 4 I/Os per submit call. Note that the range covered by a submit distribution entry can be different to the range covered by the equivalent depth distribution entry.

**IO complete**

Like the above submit number, but for completions instead.

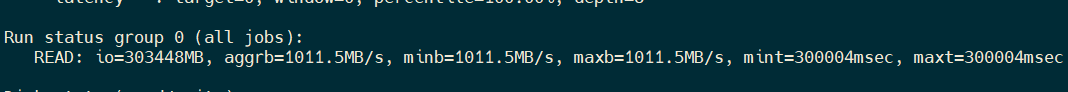
**IO issued rwt**

The number of read/write/trim requests issued, and how many of them were short or dropped.

**IO latency**

These values are for [**latency\_target**](https://fio.readthedocs.io/en/latest/fio_man.html#cmdoption-arg-latency-target) and related options. When these options are engaged, this section describes the I/O depth required to meet the specified latency target.

After each client has been listed, the group statistics are printed. They will look like this:



For each data direction it prints:

**bw**

Aggregate, minimum and maximum bandwidth of all the threads in this group. Values outside of brackets are power-of-2 format and those within are the equivalent value in a power-of-10 format.

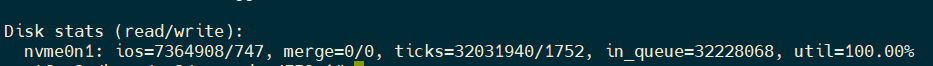
**io**

Aggregate I/O performed of all threads in this group. The format is the same as bw.

**run**

The smallest and longest runtimes of the threads in this group.

And finally, the disk statistics are printed. This is Linux specific. They will look like this



Each value is printed for both reads and writes, with reads first. The numbers denote:

**ios**

Number of I/Os performed by all groups.

**merge**

Number of merges performed by the I/O scheduler.

**ticks**

Number of ticks we kept the disk busy.

**in\_queue**

Total time spent in the disk queue.

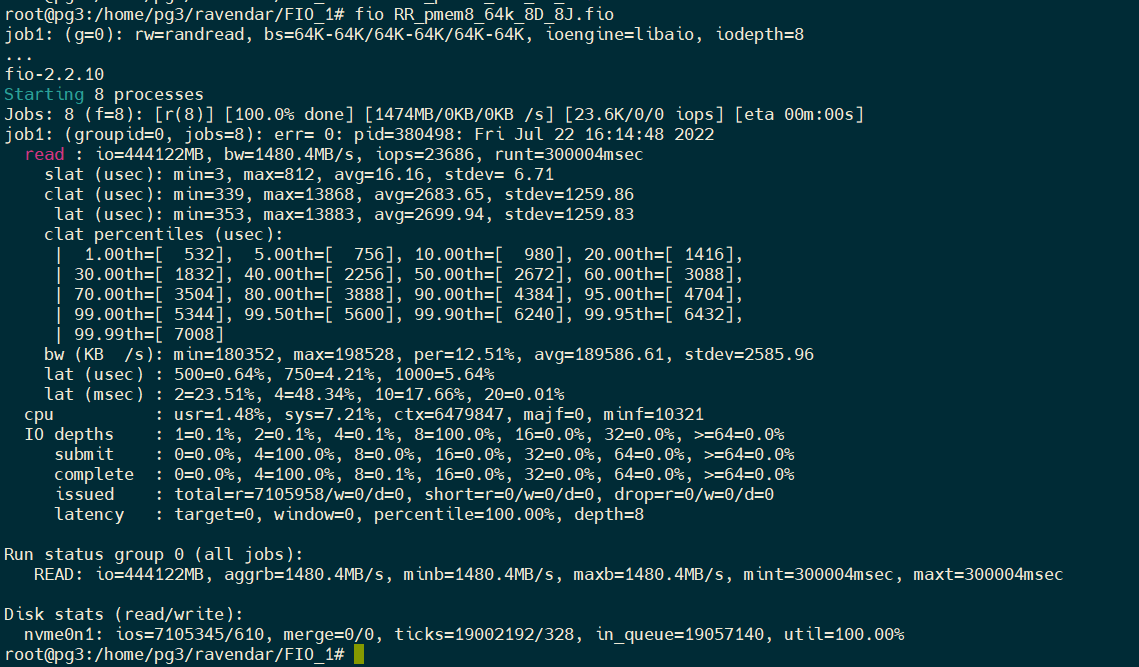
**util**

The disk utilization. A value of 100% means we kept the disk busy constantly, 50% would be a disk idling half of the time.

**Test Run for Random read**:

|  |
| --- |
| [global]  time\_based  group\_reporting  norandommap  ioengine=libaio  direct=1  buffered=0  size=10g  randrepeat=0  refill\_buffers  numjobs=8  rw=randread  iodepth=8  bs=64k  runtime=300  [job1]  #directory=/mnt/pmem8  directory=/mnt/pmem |

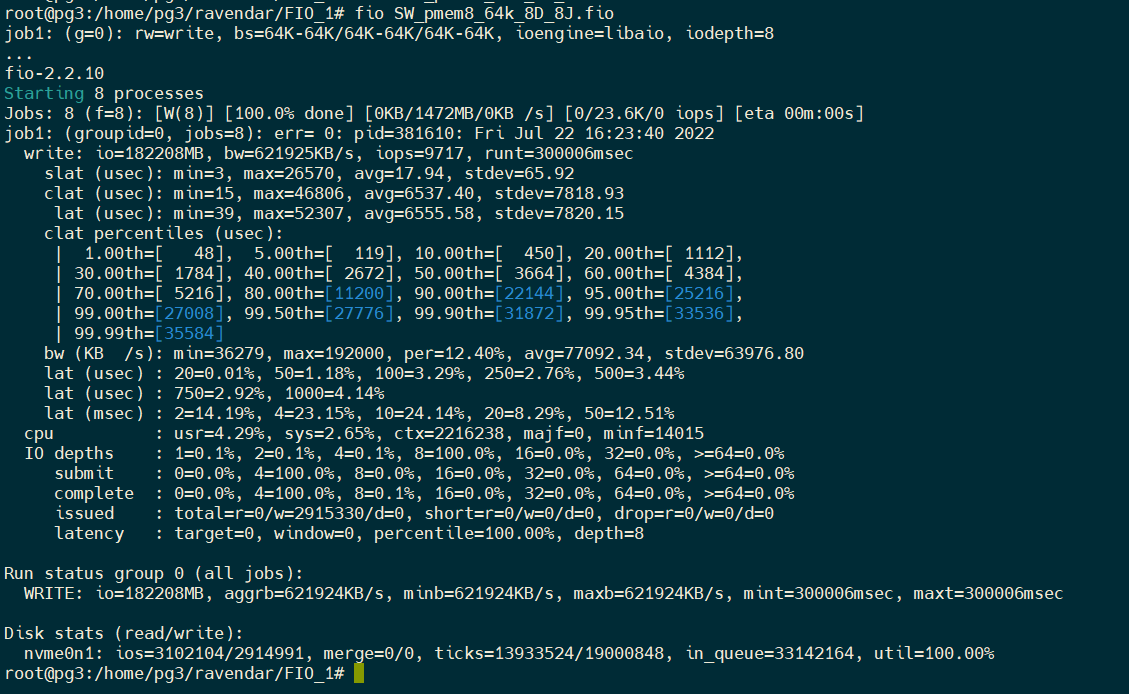
OUTPUT:



**Test Run for Sequential write:**

|  |
| --- |
| [global]  time\_based  group\_reporting  norandommap  ioengine=libaio  direct=1  buffered=0  size=1g  randrepeat=0  refill\_buffers  numjobs=8  rw=write  iodepth=8  bs=64k  runtime=300  [job1]  #directory=/mnt/pmem8  directory=/mnt/pmem |

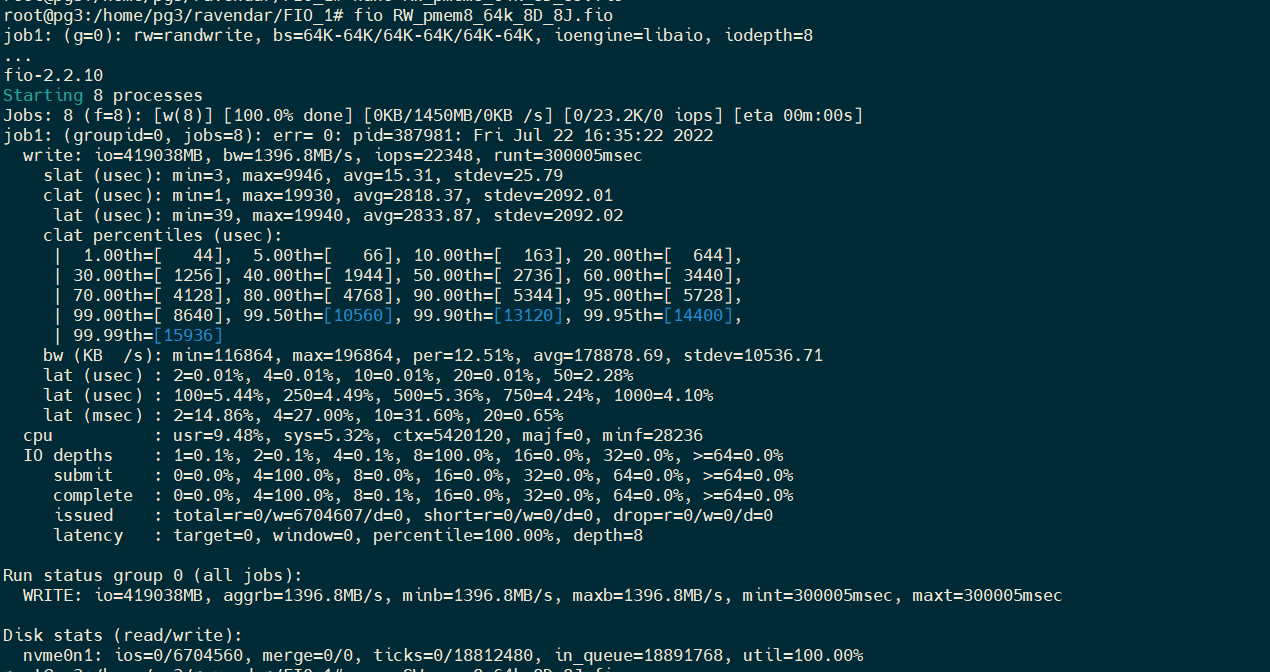
OUTPUT:



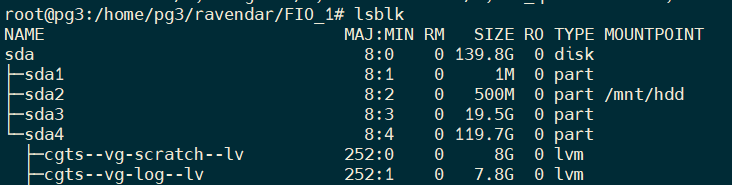
**Test Run for Random Write:**

|  |
| --- |
| [global]  time\_based  group\_reporting  norandommap  ioengine=libaio  direct=1  buffered=0  size=1g  randrepeat=0  refill\_buffers  numjobs=8  rw=randwrite  iodepth=8  bs=64k  runtime=300  [job1]  #directory=/mnt/pmem8  directory=/mnt/pmem |

Output:



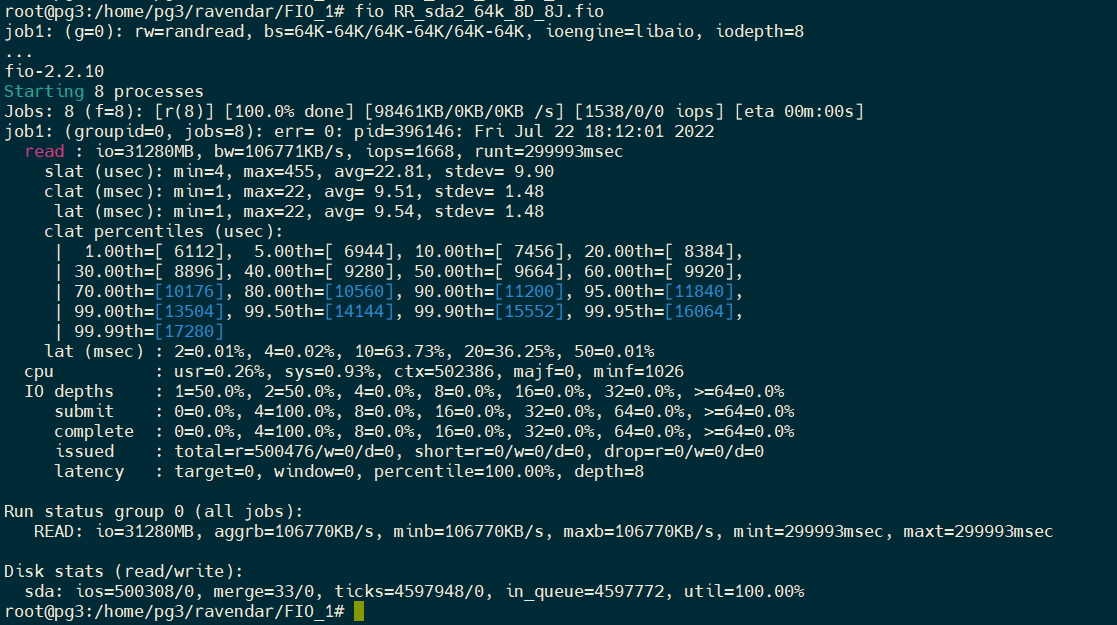
**Now lets mount some other disks:**



**Test Run for Random read**:

|  |
| --- |
| [global]  time\_based  group\_reporting  norandommap  ioengine=libaio  direct=1  buffered=0  size=128k  randrepeat=0  refill\_buffers  numjobs=8  rw=randread  iodepth=8  bs=64k  runtime=300  [job1]  #directory=/mnt/hdd  directory=/mnt/hdd |

Output:



**Test Run for Sequential read**:

|  |
| --- |
| [global]  time\_based  group\_reporting  norandommap  ioengine=libaio  direct=1  buffered=0  size=128k  randrepeat=0  refill\_buffers  numjobs=8  rw=read  iodepth=8  bs=64k  runtime=300  [job1]  #directory=/mnt/hdd  directory=/mnt/hdd |

Output:

