fio Disk I/O benchmark

START HERE

`fio --name=t --filename=<file> --size=1g --rw=randread --bs=4k`
Random read test

`fio <jobfile>`
Run a job file

`--ioengine=libaio --direct=1`
Bypass page cache

`--iodepth=<n>`
Queue depth

`--runtime=<s> --time_based`
Run for fixed time

`--numjobs=<n> --group_reporting`
Parallel workers

`--output-format=json`
Machine output

`fio --showcmd <jobfile>`
Job file as CLI

CONCEPTS

Writing to a raw device (/dev/sdX, /dev/nvme0n1) destroys its data. Benchmark a file on the filesystem, or a scratch device you can wipe.

Job file / CLI workload
↓ parsed
Jobs numjobs clones
↓ submit
ioengine iodepth in flight
↓ complete
Report IOPS, BW, latency
One fio run

RUNNING

`fio <jobfile>`
Run a job file

`fio --name=<job> --rw=<mode> ...`
Define a job on the CLI

`fio --section=<name> <jobfile>`
Run only one section

`fio --output=<file> --output-format=<fmt>`
Write report; normal/terse/json/json+

`fio --minimal <jobfile>`
Terse semicolon-separated output

`fio --showcmd <jobfile>`
Convert job file to CLI

`fio --parse-only <jobfile>`
Check options, no I/O

`fio --readonly ...`
Refuse write jobs
A safety net when pointing at real devices.

`fio --cmdhelp=<option>`
Help for one option

`fio --enghelp`
List ioengines

`fio --eta-newline=<time>`
One progress line per interval

`fio --status-interval=<time>`
Print stats periodically

`fio --version`
Show version

ACCESS PATTERN

`--rw=read / write`
Sequential read / write

`--rw=randread / randwrite`
Random read / write

`--rw=randrw --rwmixread=<pct>`
Mixed random I/O

`--rw=rw`
Mixed sequential I/O

ACCESS PATTERN (cont.)

`--bs=<size>`
Block size, e.g. 4k or 1m

`--bs=<r>,<w>`
Separate read,write sizes

`--bsrange=<lo>-<hi>`
Random block size range

`--bssplit=<size>/<pct>:...`
Weighted block sizes

`--percentage_random=<pct>`
Share of random I/O

`--norandommap`
Allow repeated blocks

`--randseed=<n>`
Repeatable random sequence

`--rw=trim`
Discard blocks

SIZE AND FILES

`--filename=<path>`
File or device to test

`--directory=<dir>`
Place generated files here

`--size=<size>`
I/O region per job

`--io_size=<size>`
Total I/O to do

`--offset=<size>`
Start offset in target

`--nrfiles=<n>`
Files per job

`--filesize=<lo>-<hi>`
Random file sizes

`--loops=<n>`
Repeat the job

`--unlink=1`
Delete files when done

ENGINE AND QUEUE

`--ioengine=libaio`
Linux async I/O

`--ioengine=io_uring`
Modern Linux async I/O

`--ioengine=psync`
Plain pread/pwrite (default)

`--ioengine=sync`
Basic read/write

`--ioengine=posixaio`
POSIX async I/O

`--ioengine=null`
No I/O, test fio itself

`--iodepth=<n>`
I/Os kept in flight
Has little effect on synchronous engines.

`--iodepth_batch_submit=<n>`
Submit in batches

ENGINE AND QUEUE (cont.)

`--direct=1`
O_DIRECT, skip page cache

`--buffered=0`
Same as direct=1

`--sync=1`
Open with O_SYNC

`--fsync=<n>`
fsync every N writes

`--end_fsync=1`
fsync at job end

`--invalidate=0`
Keep cached pages

`--hipri`
Polled completions

`--fixedbufs --registerfiles`
io_uring fast path

TIME AND LOAD

`--runtime=<time>`
Stop after this long

`--time_based`
Loop until runtime ends

`--ramp_time=<time>`
Warm-up excluded from stats

`--numjobs=<n>`
Parallel clones of the job

`--group_reporting`
One summary for all clones

`--rate=<bw>`
Cap bandwidth, e.g. 10m

`--rate_iops=<n>`
Cap IOPS

`--thinktime=<us>`
Pause after each I/O

`--startdelay=<time>`
Delay job start

`--cpus_allowed=<list>`
Pin to CPUs, e.g. 0-3

`--latency_target=<time>`
Find max load under latency

JOB FILES

Job file layout

```
[global]
ioengine=libaio
direct=1
runtime=60
time_based

[randread-4k]
rw=randread
bs=4k
iodepth=32
```

`[global]`
Defaults for all jobs

`stonewall`
Wait for earlier jobs

JOB FILES (cont.)

`--wait_for=<job>`
Start after named job

`exitall`
End all when one finishes

`${VAR}`
Environment variable in job file

`filename=<dev>:<dev>`
Several targets

OUTPUT AND LOGS

`--output-format=json`
JSON report

`--output-format=normal,json`
Several formats at once

`--write_lat_log=<prefix>`
Per-I/O latency log

`--write_bw_log=<prefix>`
Bandwidth log

`--write_iops_log=<prefix>`
IOPS log

`--log_avg_msec=<ms>`
Average log entries

`--lat_percentiles=1`
Report total latency percentiles

`--verify=crc32c --do_verify=1`
Check data read back

`--refill_buffers`
New data for every write

`--client=<host>` / `--server`
Remote run

RECIPES

4k random read IOPS

```
fio --name=randread --
filename=<file> --size=2g \
  --rw=randread --bs=4k --
ioengine=libaio --iodepth=32 \
  --direct=1 --runtime=60 --
time_based --group_reporting
```

Sequential write throughput

```
fio --name=seqwrite --
filename=<file> --size=4g \
  --rw=write --bs=1m --
ioengine=libaio --iodepth=8 \
  --direct=1 --end_fsync=1
```

70/30 mixed workload

```
fio --name=mixed --
filename=<file> --size=2g \
  --rw=randrw --rwmixread=70 --
bs=8k --iodepth=16 \
  --direct=1 --numjobs=4 --
runtime=60 --time_based \
  --group_reporting --output-
format=json --output=mixed.json
```

Latency-focused QD1 test

```
fio --name=lat --
filename=<file> --size=1g \
  --rw=randread --bs=4k --
iodepth=1 --direct=1 \
  --runtime=30 --time_based --
lat_percentiles=1
```