Virtuozzo

Virtuozzo Hybrid Infrastructure 4.7

Benchmarking Guide

October 18, 2021

Virtuozzo International GmbH

Vordergasse 59

8200 Schaffhausen

Switzerland

Tel: + 41 52 632 0411

Fax: + 41 52 672 2010

https://virtuozzo.com

Contents

CHAPTER 1

About this guide

This guide describes how to benchmark the storage system of Virtuozzo Hybrid Infrastructure using the open-source fio (Flexible I/O) tester. This benchmarking tool can generate various I/O workloads and can be run on multiple nodes in parallel. To simplify testing, a number of ready-to-use job profiles for different types of workloads are listed in the Appendix.

CHAPTER 2

Recommendations on benchmarking

Fio allows you to test storage using random and sequential workloads.

Random workloads indicate how databases, mail or web servers, and other similar software may perform if deployed inside VMs located on your storage. Such workloads typically use small blocks of data, e.g., 4K. HDDs are usually not very efficient at random reads/writes, because in such cases their read-and-write heads have to spend time on moving around the disk a lot. SSDs, in turn, typically have the block size of 1M, so they need to read and write 1M even if just 4K need to be changed. The available random read/write job profiles are set to use 4K blocks of data.

Sequential workloads can give you an idea about the performance of writing large files onto disks: creation of backups, migration of VM images, and such. Such job profiles are set to use 1M blocks of data.

Other considerations that you should take into account are:

- It is recommended to have a cluster made of identical nodes. In this case, a single fio job profile will be suitable for benchmarking all of the nodes. If your cluster nodes have different hardware, you will need to create and use as many fio job profiles as there are node hardware configurations.
- The dataset needs to be at least double the size of node's RAM. This is needed so the dataset will not fit completely into RAM and be cached, allowing you to measure disk's actual read performance.
- To improve validity of results, 3-5 iterations of the same test need to be performed.
- Each test should take at least 60 seconds, 120-300 seconds are recommended. This allows to negate the benefits of fast cache on some SSD drives.

Three main scenarios are usually tested:

- Reads: 1M sequential and 4K random.
- Writes: 1M sequential and 4K random.
- Expand: sequential writes to a file that gradually increases in size. For example, backups, database files, VM disks, etc.

CHAPTER 3

Benchmarking cluster nodes

Prepare for testing as follows:

1. Install fio 3.3 or newer on all cluster nodes and on the machine you will run the benchmarks from (unless it is one of the cluster nodes):

   ```
   # yum install fio -y
   ```

2. Create a target directory for the dataset and set its redundancy mode to 3 replicas (instead of 1 replica set by default). If you have multiple tiers, also specify the one you want to test, e.g., 1.

   ```
   # mkdir /mnt/vstorage/benchmark_dir
   # vstorage -c <cluster_name> set-attr -R /mnt/vstorage/benchmark_dir replicas=3 tier=1
   ```

   If you want to test multiple tiers, create and use a dedicated benchmark directory for every tier.

3. Download fio job profiles to a host or virtual environment that you will run the benchmark from. This host or VE must have network access to all cluster nodes.

   Note: Cluster node job profiles cannot be used to benchmark VMs.

   ---

4. On each cluster node run:

   ```
   # fio --server
   ```

   Fio will use port 8765 by default. In any case make sure that the port is open. For example, you can open it with an iptables rule:

   ```
   # iptables -I INPUT -p tcp -m tcp -m multiport --dports 8765 -j ACCEPT
   ```

5. Specify the required `size` and `numjobs` values in every node job profile:

- `size` is at least double node's RAM divided by node's CPU cores,
- `numjobs` is the number of node's CPU cores.

If Hyper-Threading is enabled, use the number of CPU threads instead. The goal is to fully load the CPU without overcommitting it.

6. Create the dataset. From the directory with the fio profiles, run

```
# for N in {<node1>,<node2>,...,<nodeN>}; do fio --minimal --client=$N prepare-set.fio; done
```

Where `nodeN` is the host name or IP address of a cluster node.

This command will prepare identical test datasets on all cluster nodes in succession. This may take several minutes but will provide less deviation of results during actual benchmarking. Ignore any performance results displayed by this fio run.

7. If cluster nodes have SSD/NVMe caches, make sure they have been flushed before running any benchmarks. You can check the caches as follows:

7.1. On a cluster node, run

```
# vstorage -c <cluster_name> top
```

7.2. While in the text-based dashboard, press **c** to expand the chunks tab and then cycle columns with **i** until you see the `JRN_FULL` and `FLAGS` columns.

7.3. Wait until `JRN_FULL` is 0% and each CS ID is marked by a **c** flag, e.g., "JC**c**". This may take some time when the cluster is not under I/O load.

Now you are ready to run the benchmark. It is recommended to run read tests before write and read/write tests.

From the directory with the fio profiles, run the benchmark on all cluster nodes as follows (`<nodeN>` is the host name or IP address of a cluster node):

- Sequential read test:

```
# fio  --client=<node1> seqread.fio --client=<node2> seqread.fio [...]
```

- Random read test:

```
# fio  --client=<node1> randread.fio --client=<node2> randread.fio [...]
```

- Sequential write test (run after making sure that caches have been flushed):

```
# fio  --client=<node1> seqwrite.fio --client=<node2> seqwrite.fio [...]
```

- Random write test (run after making sure that caches have been flushed):

```
# fio  --client=<node1> randwrite.fio --client=<node2> randwrite.fio [...]
```

- Mixed read/write test (run after making sure that caches have been flushed):

```
# fio  --client=<node1> randrw.fio --client=<node2> randrw.fio [...]
```

- Expand test:

```
# fio  --client=<node1> expand.fio --client=<node2> expand.fio [...]
```

CHAPTER 4

Benchmarking virtual machines

Aside from benchmarking cluster hardware nodes, you can also test performance from inside virtual machines created on Virtuozzo Hybrid Infrastructure. If you want to compare other virtualization solutions with Virtuozzo Hybrid Infrastructure, make sure that VMs you run tests on have identical vCPUs, RAM and other parameters.

You may want to test the following:

- A set up of one VM per cluster node. In this case, you will obtain results for the entire cluster with virtualization.
- A single VM in the cluster. In this case you will obtain the maximum performance achievable inside VMs.

To prepare for testing, do the following on any node in the compute cluster:

1. Install fio 3.3 or newer on all cluster nodes and on the machine you will run the benchmarks from (unless it is one of the cluster nodes):

   ```
   # yum install fio -y
   ```

2. Download images of test VM templates:
3. Extract VM images:

   ```
   # tar xvzf  fio-test-images.tar.gz
   ```

4. Create templates from the extracted images:

   ```
   # vinfra service compute image create --file Centos7-fio-test-sys.qcow \
   --os-distro centos7 Centos7-fio-test-sys
   # vinfra service compute image create --file Centos7-fio-test-blank.qcow \
   Centos7-fio-test-blank
   ```

5. Deploy test VMs from templates:

```
# vinfra service compute server create FIO-Centos7 --count <N> --network id=<net> \
--volume source=image,id=Centos7-fio-test-sys,size=10,boot-index=0 \
--volume source=image,id=Centos7-fio-test-blank,size=64,boot-index=1 --flavor large
```

Where

- <N> is the number of nodes in your compute cluster, e.g., 5. It is recommended to create VMs on a clean cluster, so that each node will have one VM. If you want to test a single VM, use `--count 1`.
- <net> is a network accessible by the host or virtual environment you will run the tests from, e.g., "Public".

This command will create and run the set number of identical virtual machines with these characteristics:

- 4 vCPUs and 8 GB RAM (flavor "Large"). If you want to use another flavor, make sure to change job profiles accordingly: set `size` to at least double VM's RAM divided by vCPUs and `numjobs` to the number of vCPUs.
- A 10 GB Centos 7 system disk and a 64 GB ext4 disk for tests.
- Attached to the chosen network.
- Login: `root`, password: `CDEkOA%rXfd%wWnOHDXm`.
- Fio benchmark installed, default fio port 8765 open, `fio --server` running.

6. Download fio job profiles to a host or virtual environment that you will run the benchmark from. This host or VE must have network access to all test VMs.

Note: VM job profiles cannot be used to benchmark cluster nodes.

---

7. Create the dataset. From the directory with the fio profiles, run

```
# for N in {<node1>,<node2>,...,<nodeN>}; do fio --minimal --client=$N prepare-set.fio; done
```

Where `nodeN` is the host name or IP address of a test VM.

This command will prepare identical test datasets on all test VMs in succession. This may take several minutes but will provide less deviation of results during actual benchmarking. Ignore any performance results displayed by this fio run.

If you only want to test a single VM per cluster, run `fio --minimal --client=<node1> prepare-set.fio`.

8. If cluster nodes have SSD/NVMe caches, make sure they have been flushed before running any benchmarks. You can check the caches as follows:

    8.1. On a cluster node, run

    ```
    # vstorage -c <cluster_name> top
    ```

    8.2. While in the text-based dashboard, press **c** to expand the chunks tab and then cycle columns with **i** until you see the `JRN_FULL` and `FLAGS` columns.

    8.3. Wait until `JRN_FULL` is 0% and each CS ID is marked by a **c** flag, e.g., "JC**c**". This may take some time when the cluster is not under I/O load.

Now you are ready to run the benchmark. It is recommended to run read tests before write and read/write tests.

From the directory with the fio profiles, run the benchmark on all test VMs as follows (`<nodeN>` is the host name or IP address of a test VM):

- Sequential read test:

  ```
  # fio  --client=<node1> seqread.fio --client=<node2> seqread.fio [...]
  ```

- Random read test:

  ```
  # fio  --client=<node1> randread.fio --client=<node2> randread.fio [...]
  ```

- Sequential write test (run after making sure that caches have been flushed):

  ```
  # fio  --client=<node1> seqwrite.fio --client=<node2> seqwrite.fio [...]
  ```

- Random write test (run after making sure that caches have been flushed):

  ```
  # fio  --client=<node1> randwrite.fio --client=<node2> randwrite.fio [...]
  ```

- Mixed read/write test (run after making sure that caches have been flushed):

  ```
  # fio  --client=<node1> randrw.fio --client=<node2> randrw.fio [...]
  ```

- Expand test:

  ```
  # fio  --client=<node1> expand.fio --client=<node2> expand.fio [...]
  ```

CHAPTER 5

Understanding results

Fio will start displaying results after you execute the benchmarking command. The more nodes participate in the test, the more results will be output by fio. However, a summary will be provided at the end indicated by the words “All clients”. For example:

```
All clients: (groupid=0, jobs=80): err= 0: pid=0: Mon Jan 28 19:18:23 2019
 read: IOPS=198k, BW=772Mi (809M)(45.3GiB/60078msec)
  slat (usec): min=2, max=1510, avg= 6.77, stdev= 3.29
  clat (nsec): min=1567, max=287148k, avg=3787988.85, stdev=6333559.68
   lat (usec): min=50, max=287152, avg=3794.83, stdev=6333.46
 bw (  KiB/s): min= 2048, max=113536, per=0.08%, avg=9880.44, stdev=4052.50, samples=9600
 iops        : min=  512, max=28384, avg=2470.06, stdev=1013.12, samples=9600
write: IOPS=84.8k, BW=331Mi (347M)(19.4GiB/60078msec)
  slat (usec): min=2, max=1055, avg= 7.21, stdev= 3.33
  clat (usec): min=138, max=498280, avg=21347.45, stdev=40779.33
   lat (usec): min=144, max=498296, avg=21354.72, stdev=40779.01
 bw (  KiB/s): min=  968, max=49824, per=0.08%, avg=4240.17, stdev=1738.29, samples=9600
 iops        : min=  242, max=12456, avg=1059.99, stdev=434.58, samples=9600
lat (usec)   : 2=0.01%, 20=0.01%, 50=0.01%, 100=0.71%, 250=9.39%
lat (usec)   : 500=7.81%, 750=3.96%, 1000=3.80%
lat (msec)   : 2=12.40%, 4=20.13%, 10=24.61%, 20=9.37%, 50=4.52%
lat (msec)   : 100=1.43%, 250=1.70%, 500=0.17%
cpu          : usr=0.68%, sys=3.44%, ctx=15563500, majf=0, minf=491
IO depths    : 1=0.1%, 2=0.1%, 4=0.1%, 8=0.1%, 16=0.1%, 32=100.0%, >=64=0.0%
   submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
   complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.1%, 64=0.0%, >=64=0.0%
   issued rwt: total=11866050,5093585,0, short=0,0,0, dropped=0,0,0
```

This is a sample output from a `randrw.fio` (random read/write) job run on an all-flash cluster of five nodes. The main metrics are IOPS (`iops`) and bandwidth (`bw`):

```
read: IOPS=198k, BW=772Mi (809M)(45.3GiB/60078msec)
write: IOPS=84.8k, BW=331Mi (347M)(19.4GiB/60078msec)
```

Pay attention to IOPS when you benchmark random workloads and observe bandwidth when you test

sequential operations. When you test random reads/writes also note the `iodepth` parameter that indicates how many requests are queued and waiting.

In addition:

- `slat` is submission latency that indicates how much time it took to submit I/O to the kernel.
- `clat` is completion latency that indicates how much time passed between submission to the kernel and completing the I/O (excluding submission latency).
- `lat` is the sum of `slat` and `clat`.

CHAPTER 6

Appendix: Job profile listings

The following sections list job profiles that are made available to simplify benchmarking. Each profile contains a number of parameters (on the example of `randread.fio`):

rw=randread

Fio scenario. In this case, random reads.

size=8g

Dataset size. Needs to be at least twice the size of RAM.

numjobs=16

Number of parallel tasks. Needs to be equal to the number of CPU cores.

bs=4k

Data block size.

ioengine=libaio

Asynchronous library. Requires the parameter 'direct=1'.

iodepth=32

Queue depth for asynchronous requests.

time_based

Instructs fio to loop the scenario for the duration of 'runtime' if it completes sooner or end the scenario if the time runs out.

runtime=60

Scenario duration (in seconds by default).

direct=1

This parameter forces data to be written directly to disk, without caching, and allow you to measure

disk's actual write performance.

filename_format=__testfile'$jobnum'

File name format. In this case it depends on the number of tasks.

thread

Task type. If not set, processes will be created instead of threads.

directory=/mnt/vstorage/benchmark_dir

Target directory for the dataset.

6.1 randread.fio

Tests random read performance.

```
[randread]
rw=randread
size=2x<RAM>/<CPU_cores>
numjobs=<CPU_cores>
bs=4k
ioengine=libaio
iodepth=32
time_based
runtime=60
direct=1
filename_format=__testfile'$jobnum'
thread
directory=/mnt/vstorage/benchmark_dir
```

```
[randread]
rw=randread
size=4g
numjobs=4
bs=4k
ioengine=libaio
iodepth=32
time_based
runtime=60
direct=1
filename_format=__testfile'$jobnum'
thread
directory=/sdb/
```

6.2 randwrite.fio

Tests random write performance.

```
[randwrite]
rw=randwrite
size=2x<RAM>/<CPU_cores>
numjobs=<CPU_cores>
bs=4k
ioengine=libaio
iodepth=32
time_based
runtime=60
direct=1
filename_format=__testfile'$jobnum'
thread
directory=/mnt/vstorage/benchmark_dir
```

```
[randwrite]
rw=randwrite
size=4g
numjobs=4
bs=4k
ioengine=libaio
iodepth=32
time_based
runtime=60
direct=1
filename_format=__testfile'$jobnum'
thread
directory=/sdb/
```

6.3 randrw.fio

Tests random read/write performance.

```
[randrw]
rw=randrw
rwmixread=70
rwmixwrite=30
size=2x<RAM>/<CPU_cores>
numjobs=<CPU_cores>
bs=4k
ioengine=libaio
iodepth=32
time_based
runtime=60
```

```
direct=1
filename_format=__testfile'$jobnum'
thread
directory=/mnt/vstorage/benchmark_dir
```

```
[randrw]
rw=randrw
rwmixread=70
rwmixwrite=30
size=4g
numjobs=4
bs=4k
ioengine=libaio
iodepth=32
time_based
runtime=60
direct=1
filename_format=__testfile'$jobnum'
thread
directory=/sdb/
```

6.4 seqread.fio

Tests sequential read performance.

```
[seqread]
rw=read
size=2x<RAM>/<CPU_cores>
numjobs=<CPU_cores>
bs=1m
ioengine=libaio
iodepth=32
time_based
runtime=60
direct=1
filename_format=__testfile'$jobnum'
thread
directory=/mnt/vstorage/benchmark_dir
```

```
[seqread]
rw=read
size=4g
numjobs=4
bs=1m
ioengine=libaio
iodepth=32
time_based
runtime=60
```

```
direct=1
filename_format=__testfile'$jobnum'
thread
directory=/sdb/
```

6.5 seqwrite.fio

Tests sequential write performance.

```
[seqwrite]
rw=write
size=2x<RAM>/<CPU_cores>
numjobs=<CPU_cores>
bs=1m
ioengine=libaio
iodepth=32
time_based
runtime=60
direct=1
filename_format=__testfile'$jobnum'
thread
directory=/mnt/vstorage/benchmark_dir
```

```
[seqwrite]
rw=write
size=4g
numjobs=4
bs=1m
ioengine=libaio
iodepth=32
time_based
runtime=60
direct=1
filename_format=__testfile'$jobnum'
thread
directory=/sdb/
```

6.6 expand.fio

Tests sequential writes to a gradually growing file.

```
[expand]
rw=write
size=2x<RAM>/<CPU_cores>
numjobs=<CPU_cores>
```

```
bs=16m
ioengine=sync
iodepth=32
fallocate=0
time_based
runtime=60
direct=1
filename_format=__testfile'$jobnum'
thread
directory=/mnt/vstorage/benchmark_dir
```

```
[expand]
rw=write
size=4g
numjobs=4
bs=16m
ioengine=sync
iodepth=32
fallocate=0
time_based
runtime=60
direct=1
filename_format=__testfile'$jobnum'
thread
directory=/sdb/
```

6.7 prepare-set.fio

Creates a dataset for tests.

```
[seqwrite]
rw=write
size=2x<RAM>/<CPU_cores>
numjobs=<CPU_cores>
bs=1m
ioengine=libaio
iodepth=32
time_based
#runtime=60
direct=1
filename_format=__testfile'$jobnum'
thread
directory=/mnt/vstorage/benchmark_dir
```

```
[seqwrite]
rw=write
size=4g
numjobs=4
```

```
bs=1m
ioengine=libaio
iodepth=32
time_based
#runtime=60
direct=1
filename_format=__testfile'$jobnum'
thread
directory=/sdb/
```