Virtuozzo

Virtuozzo Hybrid Infrastructure 5.3

Benchmarking Guide

12/13/2022

Table of contents

Introduction

This guide describes how to set up the storage cluster of Virtuozzo Hybrid Infrastructure, to run performance benchmarks for the network and storage disks, as well as for the NFS, iSCSI, and S3 protocols.

We recommend benchmarking the infrastructure to ensure that the measured performance matches your expectations. Benchmarking is impossible while the system is used for production workloads, as benchmarks might disrupt user activities or significantly decrease performance. You can perform the benchmarks immediately as soon as the cluster is deployed and configured. At this stage, most performance issues and misconfigurations can be detected and prevented.

The specifics of every permutation of network environment, hardware platform, and workload cannot be covered in every detail. This guide offers generic instructions on setting up a test environment, so your mileage may vary. For more instructions, refer to the Administrator Guide.

Storage cluster best practices

Using similar hardware

All cluster nodes must have identical or very similar hardware. Otherwise, the cluster will be imbalanced in terms of performance, because the cluster performance is limited by the slowest node in the cluster. This includes CPU, the amount of RAM, network cards, storage devices, controllers, etc.

Moreover, all of the disks that are assigned to the same storage tier and role must be identical in technology and size. For example, if you have different HDD disks in the same storage tier, it will lead to unpredictable performance results, because the cluster speed is constrained by the slowest device in the tier. Also, with HDDs of a different size, the physical storage space is used inefficiently, resulting in unused resources.

Enabling NVMe performance

Enable NVMe performance to boost the performance of very fast devices such as NVMes. For details on enabling and configuring this feature, refer to the Administrator Guide.

External caching

If your cluster has HDD disks, some workloads might benefit from an additional caching layer of fast devices, such as SSDs or NVMes. Keep in mind that such a configuration is only optimal for some workloads, but when possible, the gained performance completely justifies the added costs. For details on storage cache configuration, refer to the Administrator Guide.

Enabling RDMA

If your cluster is equipped with RDMA-capable network cards (the only supported model is Mellanox ConnectX-5), then you can enable RDMA for the storage backend network. This will reduce network latency, especially with random workloads.

Note that it is usually preferable to enable RDMA before creating the storage cluster.

Using jumbo frames

If your cluster has 10+ Gbit/s network adapters, you can configure them to use jumbo frames (9000-byte MTU), to achieve full performance.

To avoid performance issues, ensure that all the endpoints involved have the same MTU values.

Note that jumbo frames are currently not supported when using the Intel ixgbe network driver.

Choosing a cluster size

When choosing between building one large cluster and multiple small clusters, with an equal number of nodes in total, it is always preferable to have a single large cluster. The larger the cluster, the better its performance, efficiency, and redundancy.

Benchmarking the network

To benchmark the infrastructure network, we test network connectivity between all of the cluster nodes, and then the overall network throughput under the maximum load.

Testing network connectivity between nodes

Start the iperf3 server on each node:

```
# iperf3 -s -p 30001
```

Start the benchmark on each node:

```
# NODES=`vinfra node list -f json | jq -r '.[].host' | grep -v $(hostname -s)`
# for i in $NODES; do iperf3 -c $i -p 30001 ; done | egrep -i 'local|sender|receiver'
```

The output will be similar to the following:

```
[  4] local 172.24.4.1 port 40666 connected to 172.24.4.2 port 30001
[  4]   0.00-10.00  sec  28.6 GBytes  24.5 Gbits/sec   44             sender
[  4]   0.00-10.00  sec  28.6 GBytes  24.5 Gbits/sec                  receiver
[  4] local 172.24.4.1 port 44968 connected to 172.24.4.4 port 30001
[  4]   0.00-10.00  sec  22.1 GBytes  19.0 Gbits/sec  1672            sender
[  4]   0.00-10.00  sec  22.1 GBytes  19.0 Gbits/sec                  receiver
```

Testing network throughput under load

Start one iperf3 server per client on each node:

```
# for i in 1 2 3; do iperf3 -p 3000$i -s > /dev/null 2>&1 &  done
```

Start the benchmark simultaneously on all of the nodes:

```
# PORT=30001 # use a different port on every client (ranged 30001…30003)
# NODES=`vinfra node list -f json | jq -r '.[].host' | grep -v $(hostname -s )`
# for i in $NODES; do iperf3 -c $i -p $PORT ; done | egrep -i 'local|sender|receiver'
```

Benchmarking disks

First, you need to prepare the benchmarking scripts, and then you can perform benchmarks for each disk individually and all of them cumulatively.

Preparing benchmarks for disks

Change the following parameters in the `fio` scripts listed below:

- `size` is 4G by default, but can be set to twice the node's RAM divided by the node's CPU cores.
- `numjobs` is the number of CPU cores divided by 2 for an HDD, or the number of CPU cores for an SSD.
- `iodepth` is set to 2 or 4 for an HDD, or to 32 for an SSD.

Save the following scripts with the suggested filename on each node:

`fio.write.single.randwrite`

```
[job]
blocksize=4k
rw=randwrite
direct=1
buffered=0
ioengine=libaio
iodepth=32
fdatasync=32
size=8G
numjobs=16
```

`fio.write.single.seqwrite`

```
[job]
blocksize=1m
rw=write
direct=1
buffered=0
ioengine=libaio
iodepth=32
fdatasync=32
size=8G
numjobs=16
```

`fio.write.all`

```
[job]
blocksize=4k
rw=randwrite
direct=1
buffered=0
ioengine=libaio
iodepth=32
```

```
fdatasync=32
size=8G
numjobs=16
```

fio.read.random

```
[job]
blocksize=4k
rw=randread
direct=1
buffered=0
ioengine=libaio
iodepth=32
fdatasync=32
size=8g
numjobs=16
```

fio.write.random

```
[job]
blocksize=4k
rw=randwrite
direct=1
buffered=0
ioengine=libaio
iodepth=32
fdatasync=32
size=8g
numjobs=16
```

fio.read.seq

```
[job]
blocksize=1m
rw=read
direct=1
buffered=0
ioengine=libaio
iodepth=16
fdatasync=32
size=8g
numjobs=16
```

fio.write.seq

```
[job]
blocksize=1m
rw=write
direct=1
buffered=0
ioengine=libaio
iodepth=16
```

```
fdatasync=32
size=8g
numjobs=16
```

cs_write_test.sh

```
#!/bin/sh

cs_mounts=$(vstorage -c $(cat /mnt/vstorage/.vstorage.info/clustername) list-services |
grep vstorage.*cs | awk '{print$NF}' )
fio_config="$1"

for d in $cs_mounts; do
        cs=`cat $d/control/id`
        echo "Testing $cs in $d..."
        test_dir="$(dirname $d)/test"
        mkdir $test_dir
        f=$test_dir/file
        log_file="$cs.$fio_config.log"
        fio --group_reporting --filename=$f $fio_config > $log_file
        result=`grep "write:" $log_file`
        echo $result
        rm -f $f
        rmdir $test_dir
done
```

cs_all_write_test.sh

```
#!/bin/sh

cs_mounts=$(vstorage -c $(cat /mnt/vstorage/.vstorage.info/clustername) list-services |
grep vstorage.*cs | awk '{print$NF}' )
fio_config="$1"

echo "Test drives simlultaneously..."
filenames=""
log_file="$fio_config.log"

for d in $cs_mounts; do
        cs=`cat $d/control/id`
        test_dir="$(dirname $d)/test"
        mkdir $test_dir
        f=$test_dir/file
        path="--filename=$f"
        filenames="$filenames $path"
done

fio --group_reporting $filenames $fio_config > $log_file
result=`grep "write:" $log_file`
echo $result

for d in $cs_mounts; do
```

```
        test_dir="$(dirname $d)/test"
        f=$test_dir/file
#       echo "Cleaning $f ..."
        rm -f $f
        rmdir $test_dir
done
```

storage_test.sh

```
#!/bin/sh

fio_config="$1"
log_file="$(hostname).$fio_config.log"

test_dir="/mnt/vstorage/test/$(hostname)"
f="$test_dir/file"

mkdir -p $test_dir
echo "Testing in $f ..."
fio --group_reporting --filename=$f $fio_config > $log_file

result=`egrep "read:|write:" $log_file`
echo $result

rm -f $f
rmdir $test_dir
```

Running storage benchmarks

1. On each node, install the `fio` utility:

   ```
   # yum install fio
   ```

2. On each node, test the random and sequential write IOPS performance of each drive:

   ```
   # ./cs_write_test.sh fio.write.single.randwrite
   # ./cs_write_test.sh fio.write.single.seqwrite
   ```

3. On any node, check the cumulative random and sequential write IOPS performance:

   ```
   # grep 'write:' *fio.write.single.randwrite.log | awk -F'[=k,]' '{sum+=$2} END {print
   sum}'
   # grep 'write:' *fio.write.single.seqwrite.log | awk -F'[=k,]' '{sum+=$2} END {print
   sum}'
   ```

4. On each node, test the cumulative random write IOPS:

   ```
   # ./cs_all_write_test.sh fio.write.all
   ```

You can also test the desired replication settings as follows:

1. 1. Create the test directory `/mnt/vstorage/test`, and then apply the desired replication settings on it. For example:

```
# mkdir /mnt/vstorage/test
# vstorage set-attr -R /mnt/vstorage/test replicas=3:2
# vstorage set-attr -R /mnt/vstorage/test failure-domain=host
# vstorage set-attr -R /mnt/vstorage/test tier=0
```

2. On any node, test the random read and write IOPS performance:

```
# ./storage_test.sh fio.read.random
# ./storage_test.sh fio.write.random
```

3. On any node, test the sequential read and write IOPS performance:

```
# ./storage_test.sh fio.read.seq
# ./storage_test.sh fio.write.seq
```

Benchmarking NFS, iSCSI, and S3

In this guide, the open-source fio (Flexible I/O) tester is used to benchmark the performance of NFS and iSCSI, and GOSBench is used to benchmark the performance of S3 services.

To benchmark NFS, iSCSI or S3, follow this procedure:

1. Ensure that the benchmarking requirements are met.
2. [Optional] Deploy virtual machines with load generators from the prebuilt images.
3. Create the NFS, iSCSI, or S3 service in the test cluster.
4. Set up and prepare to run the benchmark.
5. Run the benchmark.

Requirements

Test cluster requirements

In order to obtain the most realistic results, the test cluster must be the same as the production cluster. If this is impossible, the test environment can be a replica of the production cluster or very similar to it.

When trying to determine the ideal setup size, you might want to plan ahead for the possibility to change the hardware configuration of your test cluster. With virtual machines, you can easily change the amount of RAM, the number of CPUs, disks, etc. However, unlike a physical environment, a virtual one might behave unpredictably.

The cluster should be made of identical nodes, to avoid performance disbalance, as the cluster performance is often limited by the performance of the slowest component.

Load generator requirements

Depending on the benchmarking goal, the number of load generators can differ:

- Use a single load generator to benchmark the maximum performance achievable by a single process.
- Use as many load generators as possible to benchmark the maximum cumulative cluster throughput.

In either case, you can run load generators as virtual machines in the Virtuozzo Hybrid Infrastructure cluster or as external machines (physical or virtual). When using external virtual machines for load generators, keep in mind that virtual machines usually share the same network infrastructure. For more accurate results, you need to make sure there is enough physical bandwidth between load generators and the test cluster, and that there are plenty of other physical resources, such as CPUs.

When targeting for the maximum cluster performance, or especially for the cumulative performance of multiple resources, we recommend planning a variable number of load generator

clients. This number can reach or exceed the number of nodes in the Virtuozzo Hybrid Infrastructure cluster. If adding more load generators increases the performance results, this means that the load generators' resources or number are not sufficient. Similarly to cluster nodes, load generators must have the same hardware configuration.

Coordinator node requirements

The coordinator node is a special node that is used to coordinate load generators, that is, to start and stop the benchmark on all other nodes. The coordinator node can be either a load generator or a cluster node.

Network requirements

We recommend providing at least 50 GiB of network bandwidth, either as a single 50 GbE link or as a bonded 2x25 GbE network connection, between load generators and the test cluster. Keep in mind that the network can be a bottleneck when measuring performance.

All nodes must be able to reach each other over a network:

- Cluster nodes must be reachable on network ports for standard iSCSI, NFS, and S3 services.
- Load generators must be reachable on TCP port 8765 for fio testing.
- The coordinator node must be reachable on TCP port 2000 for S3 testing.

You can open the ports in the admin panel or via iptables:

```
# iptables -I INPUT -p tcp -m tcp -m multiport --dports 8765 -j ACCEPT
# iptables -I INPUT -p tcp -m tcp -m multiport --dports 2000 -j ACCEPT
```

Deploying virtual machines with load generators

You can deploy virtual machines with load generators in the compute cluster of Virtuozzo Hybrid Infrastructure by using the prebuilt images.

1. Download and extract the images of the test VM templates:

   ```
   # wget https://docs.virtuozzo.com/fio/fio-test-images.tar.gz
   # tar xvzf fio-test-images.tar.gz
   ```

2. 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
   ```

3. Deploy test VMs from the 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:

- `--count <N>` is the number of nodes in your compute cluster, for example, `--count 5`. We recommend creating VMs on a clean cluster, so that each node will have one VM. If you want to test a single VM, use `--count 1`.
- `--network id=<net>` is a network accessible by the host or virtual environment that you will run the tests from, for example, `public`.

As the result, you will have a set number of identical virtual machines with the following characteristics:

- 4 vCPUs and 8 GiB RAM (the flavor `large`). If you want to use another flavor, make sure to update the fio scripts and other job profiles accordingly, as described in "Setting up the benchmark for NFS and iSCSI" (p. 14).
- A 10 GiB system disk with the CentOS 7 operating system
- A 64 GiB ext4 disk for tests
- The network specified by the `--network` option
- Login: `root`, password: `CDEkOA%rXfd%wWnOHDXm`

Creating the target storage

In the test cluster, create iSCSI, NFS, or S3 services that will serve as the target storage. To do this, follow the instructions in the Administrator Guide:

- For iSCSI targets, refer to Provisioning block storage space.
- For NFS shares, refer to Provisioning file storage space.
- For the S3 cluster, refer to Provisioning object storage space.

Note
For iSCSI and NFS, the recommended storage size is at least 3x the amount of available RAM.

Setting up the benchmark for NFS and iSCSI

To set up the benchmark for the NFS and iSCSI resources, install and confugure the fio tool, and then mount the resources to all load generators.

Installing fio

1. On the coordinator node, create a directory to store the fio scripts, for example, `/root/scripts`.

```
# mkdir /root/scripts
```


2. Download the fio benchmark scripts to the `/root/scripts` directory:

```
# cd /root/scripts
# url=https://docs.virtuozzo.com/fio/vm/; \
wget $url/expand.fio $url/prepare-set.fio $url/randread.fio $url/randrw.fio
$url/randwrite.fio $url/seqread.fio $url/seqwrite.fio
```

3. If required, install the fio benchmark tool on each load generator and on the coordinator node:

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

4. Ensure that the following parameters in the fio scripts are correct:
   - `size` is at least twice the target node's RAM, divided by the node's CPU cores. For example, for a quad-core node with 16 GiB RAM, the value should be `4g`. This parameter sets the total amount of data that each I/O thread will transfer.
   - `numjobs` is the number of 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. For example, on a quad-core node with hyper-threading, the value should be `8`. This parameter sets the amount of I/O threads that will be started.
   - `directory` is the target directory to be tested. For example, if the resource to be tested is mounted to `/mnt/test`, then update this field accordingly.

   For more options, refer to the fio documentation.

Mounting the NFS resource

Mount the NFS resource to each load generator. As a mount directory, use the target directory that you specified in the fio scripts, for example, `/mnt/test`.

To be able to mount an NFS resource, you might need to install the `nfs-utils` package.

```
# yum install nfs-utils
```

To mount the NFS resource, run:

```
# mount -t nfs -o vers=4.0 <share_IP>:<share_name> /mnt/test
```

Where:

- `-o vers=4.0` is the NFS version to use.
- `<target_IP>` is the IP address or hostname of the NFS share.
- `<share_name>` is the identifier of the NFS share.

Mounting the iSCSI resource

Access the iSCSI resource by following the detailed instructions in the Storage User Guide. Then, mount the iSCSI device to each load generator. As a mount directory, use the target directory that you specified in the fio scripts, for example, `/mnt/test`.

Setting up the benchmark for S3

To set up the benchmark for the S3 resource, install and configure the GOSBench tool as follows:

1. Download and extract the tool from GitHub on each load generator and the coordinator node:

```
# curl -OL https://github.com/mulbc/gosbench/releases/download/v0.4/gosbench_0.4_
Linux_x86_64.tar.gz
# tar -xzf gosbench_0.4_Linux_x86_64.tar.gz
```

2. On the coordinator node, create the file `gosbench_script.yaml` with the following configuration:

```
s3_config:
  - access_key: <ACCESS_KEY>
    secret_key: <SECRET_KEY>
    region: eu-central-1
    endpoint: <S3_ENDPOINT_URL>
    skipSSLverify: true

grafana_config:
  endpoint: https://<GRAFANA_ENDPOINT>
  username: admin
  password: <ADMIN_PASSWORD>

tests:
  - name: write test
    read_weight: 0
    existing_read_weight: 0
    write_weight: 100
    delete_weight: 0
    list_weight: 0
    objects:
      size_min: 1
      size_max: 100000
      part_size: 0
      size_distribution: random
      unit: KB
      number_min: 10
      number_max: 10
      number_distribution: constant
    buckets:
      number_min: 1
      number_max: 10
      number_distribution: constant
    bucket_prefix: 1255gosbenchobject_
    prefix: obj
    stop_with_runtime: 900s
    stop_with_ops:
    workers: <NUMBER_OF_WORKERS>
    workers_share_buckets: True
    parallel_clients: 3
```

```
    clean_after: True

  - name: read test
    read_weight: 100
    existing_read_weight: 0
    write_weight: 0
    delete_weight: 0
    list_weight: 0
    objects:
      size_min: 1
      size_max: 100000
      part_size: 0
      size_distribution: random
      unit: KB
      number_min: 10
      number_max: 10
      number_distribution: constant
    buckets:
      number_min: 1
      number_max: 10
      number_distribution: constant
    bucket_prefix: 1255gosbenchobject_
    prefix: obj
    stop_with_runtime: 900s
    stop_with_ops:
    workers: <NUMBER_OF_WORKERS>
    workers_share_buckets: True
    parallel_clients: 3
    clean_after: True
```

3. Update the required fields accordingly, in particular, these fields:
   - `access_key` is the S3 access key to access the resource.
   - `secret_key` is the S3 secret key to access the resource,
   - `endpoint` in the `s3_config` section is the S3 endpoint URL to access the resource.
   - `endpoint` in the `grafana_config` section is the Grafana dashboard URL, which is https://<admin_panel_IP>:8888/grafana/d/dashboard-directory-en-US/.
   - `password` is the Grafana administrator password.
   - `workers` is the number of load generators. Note that there are multiple occurrences of this parameter that need to be set to the same value.

For more details, refer to the GOSBench documentation.

Preparing to run the benchmark

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

1. On a cluster node, run:

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

2. While in the text-based dashboard, press **c** to expand the chunks tab, and then cycle columns by pressing **i** until you see the `JRN_FULL` and `FLAGS` columns.
3. Wait until `JRN_FULL` is 0% and each CS ID is marked by the `c` flag, for example, `JCc`. This may take some time when the cluster is not under I/O load. The journal cleaning can be forced by running:

```
# vstorage -c <cluster_name> set-config cs.force_checkpoint=1
# vstorage -c <cluster_name> set-config cs.force_checkpoint=0
```

Running the benchmark for NFS and iSCSI

Running fio

1. On each load generator node, start the fio server:

```
# fio --server
```

2. On the coordinator node, create the dataset. In this example, we assume the scripts are stored in the `/root/scripts` directory, and we want to run the preparation script for the three load generators at 10.10.10.11, 10.10.10.12, and 10.10.10.13:

```
# for N in {10.10.10.11,10.10.10.12,10.10.10.13}; do fio --minimal --client=$N
/root/scripts/prepare-set.fio; done
```

   You can also specify the hostnames of cluster nodes, instead if their IP addresses.

   This command will prepare identical test datasets on all cluster nodes in succession. This may take several minutes, but it will provide less deviation of results during actual benchmarking. Ignore any performance results displayed by this fio run.
3. On the coordinator node, start the desired benchmark script. In this example, we assume the scripts are stored in the `/root/scripts` directory, and we want to run `randread.fio` for the three load generators at 10.10.10.11, 10.10.10.12, and 10.10.10.13:

```
# fio --client 10.10.10.11 --client 10.10.10.12 --client 10.10.10.13
/root/scripts/randread.fio
```

Collecting fio results

Fio shows several reports, one for each load generator participating in the benchmark. At the bottom, you will find the section `"All clients"`, which contains the summary of the overall performance.

An example output is as follows:

```
[…]
All clients: (groupid=0, jobs=4): err= 0: pid=5190: Mon Mar 7 15:16:09 2022
  read: IOPS=18.9k, BW=73.9MiB/s (77.5MB/s)(8865MiB/120001msec)
    slat (usec): min=105, max=97552, avg=206.52, stdev=187.96
    clat (usec): min=5, max=299572, avg=53923.77, stdev=8760.84
     lat (usec): min=210, max=299745, avg=54131.08, stdev=8784.92
    clat percentiles (msec):
     | 1.00th=[ 44], 5.00th=[ 46], 10.00th=[ 47], 20.00th=[ 48],
     | 30.00th=[ 50], 40.00th=[ 52], 50.00th=[ 53], 60.00th=[ 54],
     | 70.00th=[ 56], 80.00th=[ 59], 90.00th=[ 64], 95.00th=[ 68],
     | 99.00th=[ 79], 99.50th=[ 82], 99.90th=[ 89], 99.95th=[ 106],
     | 99.99th=[ 296]
   bw ( KiB/s): min=39528, max=88968, per=99.97%, avg=75622.67, stdev=1440.79,
samples=957
   iops : min= 9882, max=22242, avg=18905.56, stdev=360.20, samples=957
  lat (usec) : 10=0.01%, 250=0.01%, 500=0.01%, 750=0.01%, 1000=0.01%
  lat (msec) : 2=0.01%, 4=0.01%, 10=0.01%, 20=0.01%, 50=32.91%
  lat (msec) : 100=67.02%, 250=0.01%, 500=0.04%
  cpu : usr=2.35%, sys=9.55%, ctx=2283821, majf=0, minf=1079
  IO depths : 1=0.1%, 2=0.1%, 4=0.1%, 8=0.1%, 16=0.1%, 32=0.1%, >=64=100.0%
     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=0.0%, 64=0.0%, >=64=0.1%
     issued rwts: total=2269396,0,0,0 short=0,0,0,0 dropped=0,0,0,0
     latency : target=0, window=0, percentile=100.00%, depth=256

Run status group 0 (all jobs):
   READ: bw=73.9MiB/s (77.5MB/s), 73.9MiB/s-73.9MiB/s (77.5MB/s-77.5MB/s),
io=8865MiB (9295MB),
run=120001-120001msec

Disk stats (read/write):
   sda: ios=2267039/1298, merge=0/967, ticks=417590/1434, in_queue=418444,
util=99.22%
```

In this output, some of the most relevant metrics are highlighted in bold. They include IOPS, latency (**lat**), I/O depth, and bandwidth (**bw**).

Running the benchmark for S3

Running GOSBench

Important
Ensure that you have configured the access script, as described in "Setting up the benchmark for S3" (p. 16).

1. On the coordinator node, start the benchmark server:

```
# ./server -c gosbench_script.yaml
```


2. On each load generator, start the benchmark workers:

```
# ./worker -p 8009 -s <coordinator_IP>:2000
```

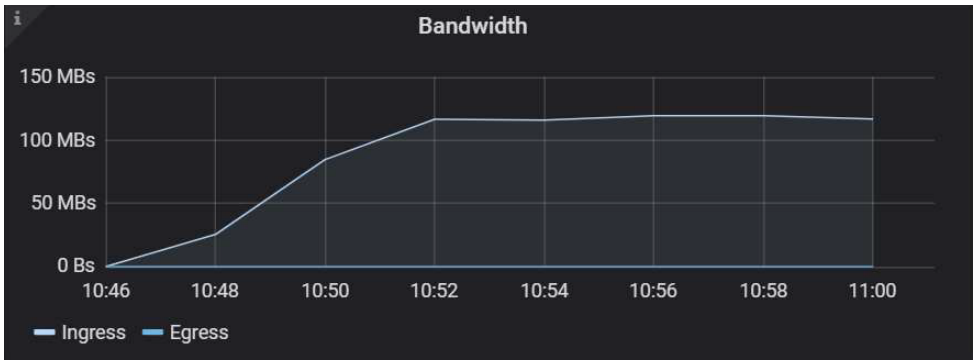
Collecting GOSBench results

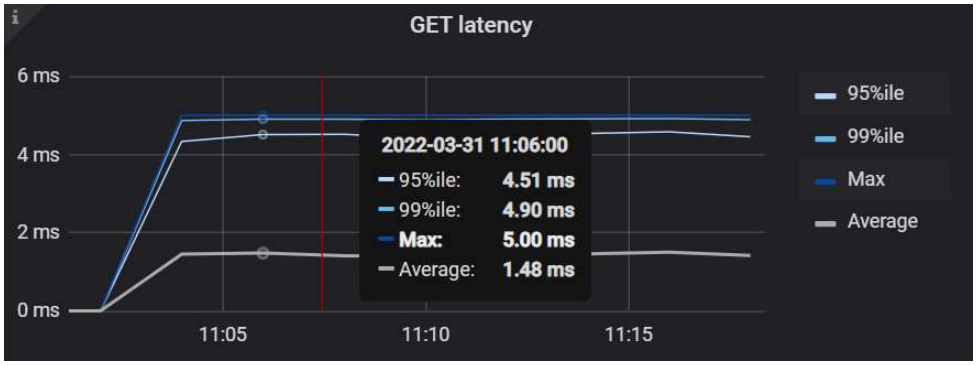
GOSBench shows performance results in the console, and also uses Prometheus and Grafana to collect and graph results.

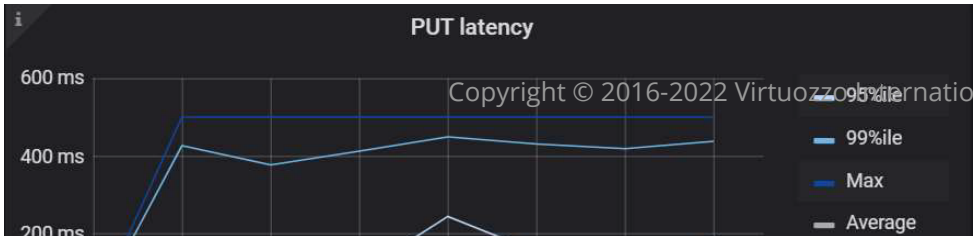
An example console output is as follows:

```
INFO[2022-06-07T15:09:47+03:00] Ready to accept connections
INFO[2022-06-07T15:09:58+03:00] 127.0.0.1:57956 connected to us
INFO[2022-06-07T15:09:58+03:00] We found worker 1 / 1 for test 0
Worker="127.0.0.1:57956"
INFO[2022-06-07T15:10:03+03:00] All workers have finished preparations - starting
performance test  test="write test"
INFO[2022-06-07T15:25:20+03:00] All workers have finished the performance test -
continuing with next test  test="write test"
INFO[2022-06-07T15:25:20+03:00] GRAFANA: ?from=1654603803922&to=1654604720745
test="write test"
INFO[2022-06-07T15:25:20+03:00] PERF RESULTS  Average BW in Byte/s=6.638758826285763e+07
\
Average latency in ms=3018.8787541713014 Test runtime on server=15m16.822798366s \
Total Bytes=6.0183302998e+10 Total Operations=899 test="write test"
```

The line starting with `INFO[<date>:<time>] GRAFANA` includes a query string that you can append to the Grafana dashboard URL. The admin panel provides the Grafana server and the **S3 overview** dashboard that you can use to check the benchmark behavior:







General considerations

- To improve the validity of results, you need to perform 3-5 iterations of the same test.
- For file tests, each test should take at least 60 seconds. The recommended time, however, is 120-300 seconds, which can negate the benefits of fast cache on some SSD drives.
- For S3 tests, the recommended run time is at least 900 seconds, which provides more stable results.