Virtuozzo

Virtuozzo Hybrid Infrastructure 5.4

Benchmarking and Performance Guide

9/14/2023

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, storage disks, and virtual machines, as well as for the NFS, iSCSI, and S3 protocols. It also explains how to identify and troubleshoot typical performance issues in Virtuozzo Hybrid Infrastructure.

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 will yield unreliable results and 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 information obtained with benchmarking only measures actual performance and may be not sufficient to highlight performance issues. This information helps to understand if performance is up to expectations, and can be used for comparison with monitoring tools during normal cluster activity.

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, it is strongly recommended that:

- All cluster nodes have the same number of disks in each storage tier.
- All disks assigned to the same storage tier and role are 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.

Using different hardware can also be a problem for reasons unrelated to performance. For example, using different CPUs may block migration of virtual machines between nodes.

Using the same software version

We recommend using the same software version on all cluster nodes, to avoid both performance issues and issues that may occur during maintenance operations such as adding new nodes.

Using different storage tiers for different performance goals

Storage tiers allow you to create groups of disks. As a best practice, it is preferable to group storage devices based on their technology, and group data based on its performance goals. For example, separating data with high-priority access from data with low-priority access helps to optimize data access performance for both of these data types. Generally, it is also advised to separate hot data from cold data, and replicated data from encoded data.

Keep in mind that switching from replication to erasure coding may degrade storage performance. To avoid changing the redundancy method, we recommend planning it in advance.

A typical scenario of using storage tiers is to use disks of different technology as capacity devices in the same cluster, for example, HDDs and SSDs. Also note that faster drives should be assigned to higher storage tiers. For details on storage tiers, refer to the Administrator Guide.

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

Enabling RDMA reduces network latency and improves overall throughput, especially with random workloads.

To enable RDMA, every storage node in your cluster must be equipped with RDMA-capable network cards, and the network switch must support RDMA.

Note that RDMA must be enabled before the storage cluster is created.

Using jumbo frames

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

To test if jumbo frames are working correctly, ping all other node interfaces in the storage network from each node:

```
# ping -s 8972 -M do <HOST>
```

Choosing the cluster size, redundancy, and network bandwidth

When choosing between building one large cluster and multiple small clusters, with an equal number of nodes in total, similar network latency between nodes, and with no other size limits reached (such as the limit on the number of files or chunks), it is always preferable to have a single large cluster. The larger the cluster, the better its performance, efficiency, and redundancy. A large cluster can afford to lose more nodes while still being able to heal itself automatically, avoid cluster degradation, and use more efficient erasure coding schemes.

While choosing between erasure coding schemes, you need to consider multiple factors. With the same number of parity chunks, using a higher number of data chunks increases storage efficiency, but decreases system reliability and performance.

The chosen redundancy scheme also affects the number of failure domains in a cluster. The general recommendation is to always plan at least one more failure domain in a cluster than required.

To calculate the optimal number of storage disks for each node, you need to consider network bandwidth as an upper limit of the available storage bandwidth each node can provide. The table below shows the maximum number of devices that can be hosted on each node before network bandwidth becomes a bottleneck.

Network bandwidth	Maximum number of disks per node based on device speed		
	100 MB/s	300 MB/s	1000 MB/s
10 GbE	12	4	1
2x10 GbE	21	7	2
25 GbE	31	10	3
2x 25GbE	53	17	5
50GbE	62	20	6
2x 50GbE	106	35	10
100GbE	125	41	12
2x 100GbE	212	70	21

Depending on capabilities of a network switch, its total available bandwidth can be saturated and not allow further scaling after a certain number of nodes is reached. Refer to your network switch specifications to understand if this can become a bottleneck.

Separating internal and external networks

Network bandwidth is used not only to deliver data services to network clients, but also for internode communication. Moreover, different data redundancy schemes have different network overhead. For example, when using replication with 3 replicas, serving a certain amount of write requests from applications needs at least double the amount of internal network bandwidth.

Separating internal and external data networks helps to increase available bandwidth and avoid bottlenecks.

Virtual machines performance

When trying to optimize performance of virtual machines, consider the following:

- VirtIO disks are typically more performant than disks with the default SCSI bus. Note that VirtIO disks must be thick.
- Snapshots have an impact on performance. If snapshots are not necessary, use volumes without snapshots, to improve performance.

Configuration examples

- Backups and cold storage. The aim is to maximize performance of sequential workloads. CPU, RAM, and network latency are not as important as the number of HDDs and their IOPS capabilities. Use erasure coding to decrease overall cost.
- Hot storage and compute. The aim is to minimize latency. Use NVMe disks, an RDMA network, fast CPU and RAM, and replication for better performance.
- Databases. The aim is to maximize performance of random workloads. Use NVMe or fast SSD disks, an RDMA network, and consider using no redundancy if an application has built-in replication capabilities.

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.

If the `iperf3` utility is not already installed, install it by running:

```
# yum install iperf3
```

Testing network connectivity between nodes

1. Start the `iperf3` server on each node:

   ```
   # for i in $(vinfra node list -f json | jq -r '.[].host'); do ssh $i iperf3 -s -p
   30001 -D; done
   ```

2. Start the benchmark on each node:

   ```
   # NODES=`vinfra node list -f json | jq -r '.[].host' | sort | 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
   ```

3. Terminate the `iperf` server when done.

In the test report, check the following:

- The measured network speed between all node pairs is similar, and that it is close to the rated network speed. Otherwise, this may indicate a network problem.
- The number of retries (shown in the second-to-last column) is close to zero.

Testing network throughput under load

1. Start one `iperf3` server on a different port per client on each node:

   ```
   # for i in $(vinfra node list -f json | jq -r '.[].host'); do ssh $i "for j in 1 2 3;
   \
   do iperf3 -p 3000\$j -s -D ; done" ; done
   ```

2. 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' | sort | grep -v $(hostname -s
)`
# for i in $NODES; do iperf3 -c $i -p $PORT ; done | egrep -i 'local|sender|receiver'
```

3. Terminate the `iperf` server when done.

The test tries to maximize network load by generating traffic between all node pairs, and then reports the measured network speed between pairs.

In the test report, check the following:

- The cumulative input and output of each node is close to the rated network speed.
- The number of retries (shown in the second-to-last column) is close to zero.

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

For more accurate results, we recommend tuning 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 simultaneously..."
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

These tests measure the system behavior in various simulated workload conditions. If desired, you can use them to trigger and analyze scenarios described in "Performance issues and symptoms" (p. 33).

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 virtual machines

Besides generic benchmark best practices, consider the following while running tests on virtual machines:

- Using a high number of threads or a deep queue can improve results, as virtual machine I/O tends to have more latency compared to tests running on a host.
- Using test sets with a size comparable to the size of a virtual disk, or running the same test several times, can significantly impact test results on thin disks. With each repetition a virtual disk becomes “thicker,” which leads to higher performance of subsequent tests.
- Using snapshots is generally not recommended if the goal is to maximize performance. The system needs to synchronize data of a live volume and its snapshot, which adds an I/O overhead that lowers the disk performance. The impact of snapshots, however, may be reduced by using larger test sets.

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. 20).
- 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. 22).

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
```

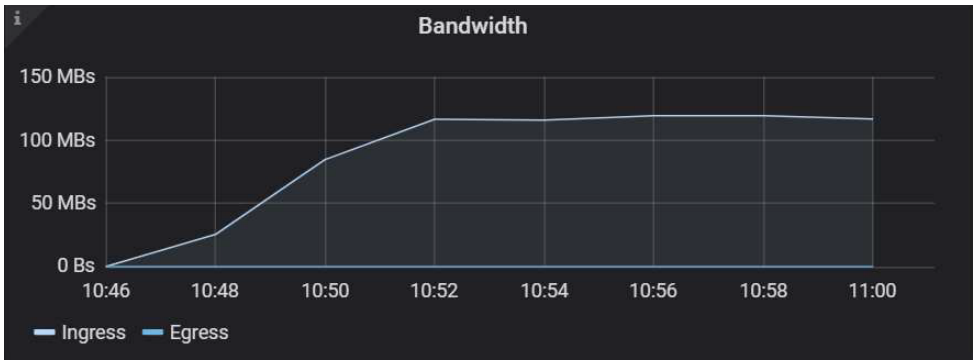
Collecting GOSBench results

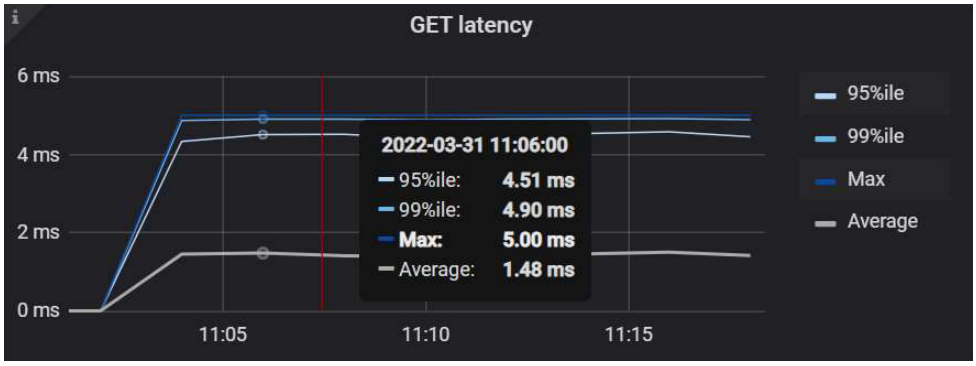
GOSBench shows performance results in the console, and also uses Prometheus and Grafana to collect and graph results.

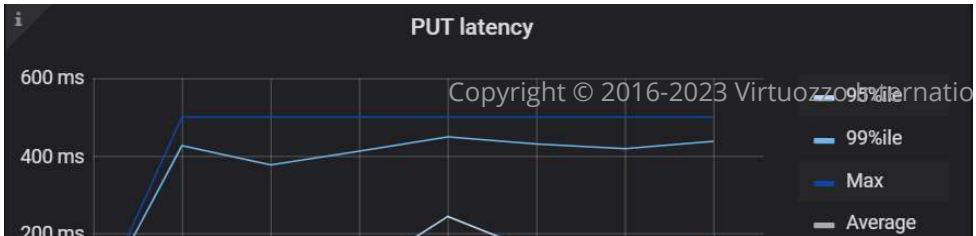
An example console output is as follows:

```
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:







Limiting performance

In some cases, it is preferable to limit the performance of the system or one of its components, usually to slow down the usage growth or avoid interference between different services.

Backup storage throttling

The backup storage service automatically starts to throttle incoming traffic when the storage usage exceeds a certain threshold, to slow down the growth rate and avoid filling the system. To tune the throttling parameters, navigate to the **Throttling configuration** tab on the backup storage **Settings** screen in the admin panel.

Re-encoding throttling

The re-encoding process is usually limited, to avoid an impact on other services. We do not recommend changing the default settings, unless re-encoding is the only service running in the cluster.

The default number of re-encoding processes is 1. To check the number of concurrent re-encoding processes, run:

```
# cat /mnt/vstorage/.vstorage.info/ls_reencoding_chunks
```

To change the number of concurrent re-encoding processes, run:

```
# echo <NUMBER> > /mnt/vstorage/.vstorage.info/ls_reencoding_chunks
```

For example:

```
# echo 4 > /mnt/vstorage/.vstorage.info/ls_reencoding_chunks
```

Troubleshooting performance issues

In Virtuozzo Hybrid Infrastructure, many performance issues and misconfigurations trigger system alerts, which are displayed in the admin panel. However, some performance issues can only be detected manually by using the command-line interface.

Performance-related system alerts

Disk is out of space

It is strongly advised to monitor the following out-of-space disk alerts closely:

- Cluster is out of space
- Disk may run out of space
- Disk is out of space
- Metadata disk is out of space

Disks at near-full capacity typically show decreased performance, especially during write operations. Filling disks further might result in other access issues, such as inability to write more data. For these reasons, we do not recommend exceeding 80 percent of used disk space at all times.

Note that this may happen regardless of space usage in an assigned storage tier. In other words, a disk may have more than 80 percent of used space even if its assigned storage tier is used by less than 80 percent.

RAID resyncing

The **Software RAID is not fully synced** alert is raised when resyncing of a software RAID device managed by the `md` service is in progress. If the MDS disk is affected, this process decreases the cluster performance until the RAID rebuilding is finished.

Too many chunks or files in the cluster

When the cluster has too many files or chunks, the following alerts are raised:

- Cluster has too many chunks
- Cluster has a critically high number of chunks
- Cluster has too many files
- Cluster has a critically high number of files

This is a sign that the cluster is reaching its resource limit, which will impact the performance of metadata operations. When this limit is reached, the performance degradation may become a problem. In this case, you can increase the chunk size (however, this may also affect the performance), or redirect some workloads to a different cluster.

The general recommendation is not to exceed four million files and ten million chunks.

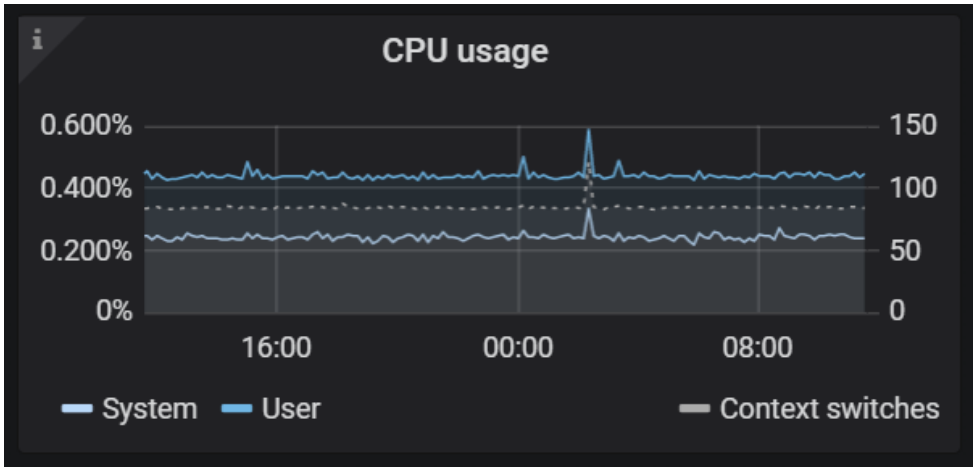
CPU overload

When the metadata (MDS) and S3 gateway services use CPU resources above a threshold, the system issues the following alerts:

- Metadata service has high CPU usage
- S3 Gateway service has high CPU usage
- S3 Gateway service has critically high CPU usage

The CPU usage of the metadata service can be checked either in Grafana or the command line.

In Grafana, you can check it on the **Virtuozzo Storage MDS details** dashboard:



To check the MDS CPU usage via the command line, use the following command:

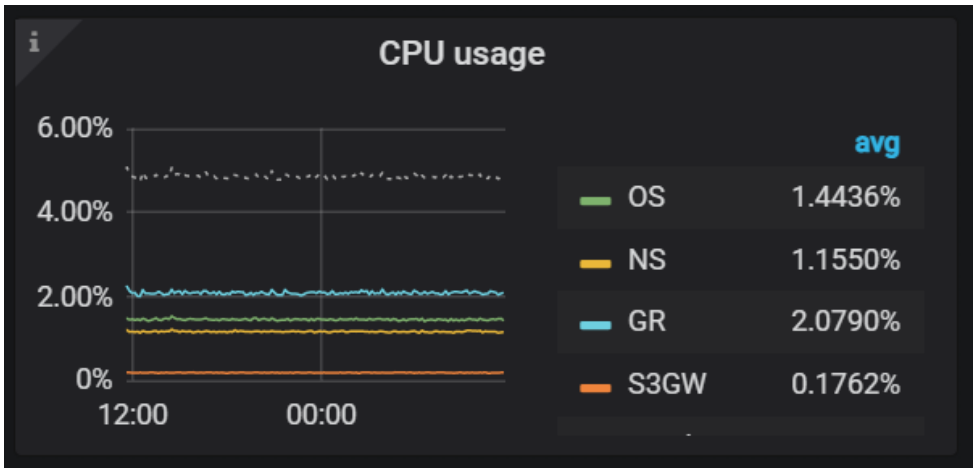
```
# vstorage -c <CLUSTER> top
```

Press **m** to focus on the metadata service. In the command output, the current CPU usage will be reported in the `%CPU` column, as highlighted in the following example:

```
MDSID STATUS   %CTIME   COMMITS   %CPU    MEM   UPTIME HOST
    3 avail      0.0%       0/s   0.2%   142m   6d  1h management…
M   1 avail      0.1%       1/s   0.2%   149m   6d  1h management…
    2 avail      0.1%       1/s   0.2%   149m   6d  1h management…
```

The alert is raised when the CPU usage is above 80 percent for at least five minutes. For the metadata service, it is normal to use 100 percent of the CPU during peak traffic. However, this may also indicate an issue if the alert persists longer than one day and it is accompanied by other performance issues (for example, high latency).

Similarly, the CPU usage of the Object storage service can be checked on the **Object Storage overview** dashboard in Grafana:



High latency

There are several processes that can be affected by high latency. The system will issue alerts when the latency of these services is too high:

- Metadata service (MDS)
- Chunk service (CS)
- Object storage (S3) services

Metadata service latency

The metadata service latency represents the response time of all metadata operations. You can check the metadata service latency on the **Virtuozzo Storage MDS details** dashboard in Grafana:

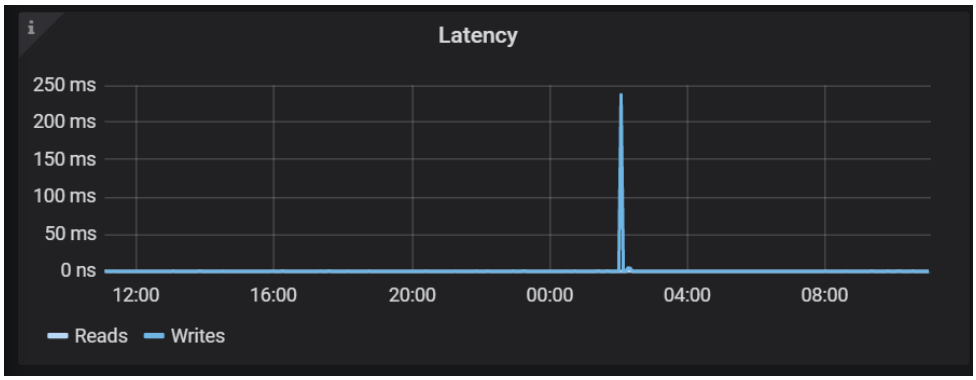
The **Metadata service has high commit latency** alert triggers when the 95th percentile latency of the service is higher than one second for more than five minutes.

Though it is considered normal for latency to increase during peak hours, it may also indicate an issue if the alert persists for more than one day and performance is below expectations.

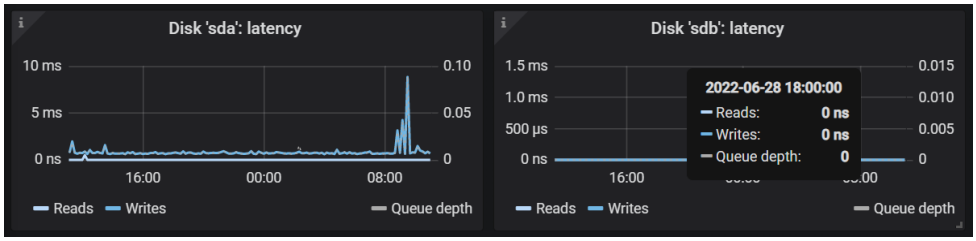
Chunk server latency

The chunk service latency can be checked either in Grafana or the command line.

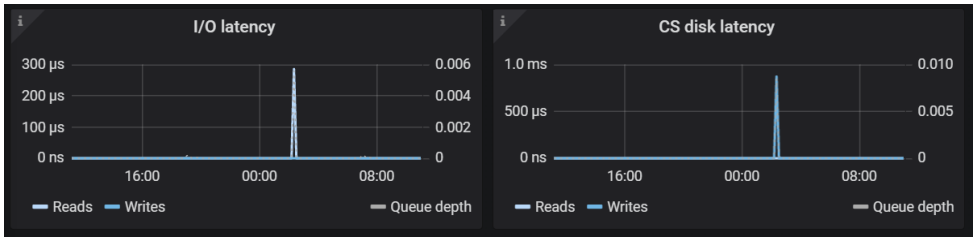
In Grafana, you can check it on the **Virtuozzo Storage core cluster overview** dashboard:



You can also monitor the latency of a particular disk on the **Hardware node details** dashboard:



Alternatively, the latency of a storage disk is shown on the **Virtuozzo Storage CS details** dashboard:



To check the chunk server latency via the command line, use the following command:

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

Press **i** to view the `optime` values. The command output will be similar to this:

```
  CSID IOWAIT  SWAIT OPTIME(ms) IOLAT(ms)  SLAT(ms) QDEPTH        RMW       JRMW
  1027     0%     0%        N/A       0/0       0/0    0.0     0ops/s     0ops/s
  1029     0%     0%        N/A       0/0       0/0    0.0     0ops/s     0ops/s
  1025     0%     0%        N/A       0/0       0/0    0.0     0ops/s     0ops/s
... (rows 1-3 of 6)
```

The `optime` represents the average time spent serving each I/O request, ignoring the time it spent in the I/O queue. If the optime is consistently high, for example, higher than 100 ms, this means that there might be an issue with the I/O path. If the issue is caused by disk wear, it can be fixed by replacing the disk. Other solutions include upgrading the device firmware, replacing faulty cables, replacing a faulty controller, or adding capacity.

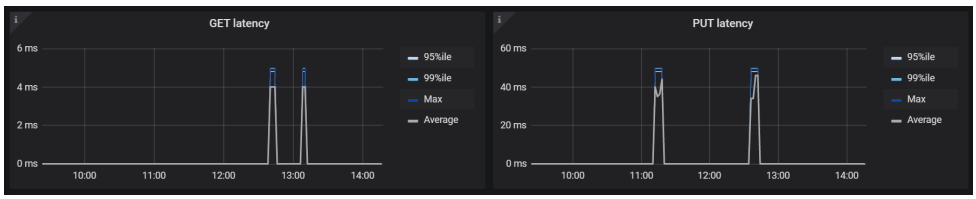
Object storage service latency

The system raises alerts when one of the following latencies exceeds a certain threshold for more than five minutes:

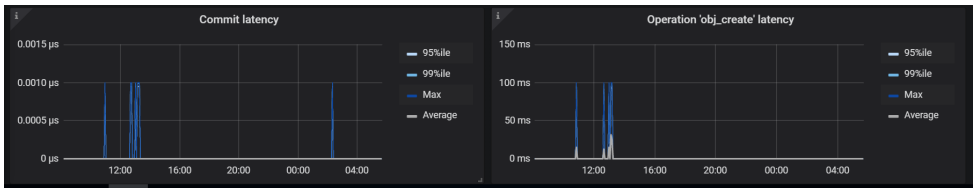
- S3 gateway GET latency
- Object service commit latency
- Object service request latency
- Name service commit latency
- Name service request latency

Though it is considered normal for latency to increase during peak hours, it may also indicate an issue if the alert persists for more than one day and performance is below expectations.

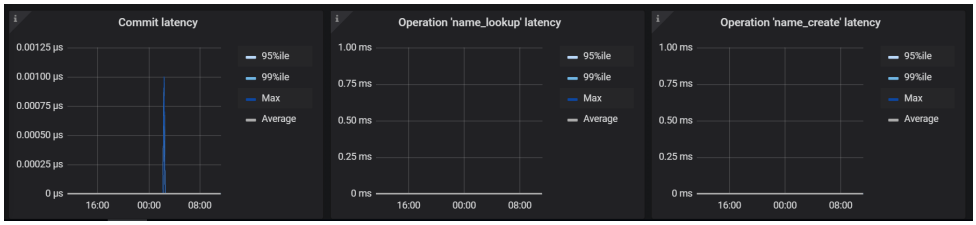
You can check the latency for the object storage services in Grafana. To monitor the S3 gateway GET and PUT latency, use the **S3 overview** dashboard:



To check the object service latency, go to the **Object Storage OS details** dashboard:



The name service latency can be checked on the **Object Storage NS details** dashboard:



S.M.A.R.T. alerts

S.M.A.R.T. alerts must be treated with high priority. All signs of disk wear can decrease the cluster performance, in addition to further performance degradation in case of a device failure (which may include the loss of redundancy and availability, and also node downtime if the system disk is affected). Moreover, recovery operations, such as storage rebalancing, may have an impact on the system performance.

Performance issues and symptoms

Disks IOPS saturation

A common root cause of performance issues is the disk throughput limit. To understand if a disk has reached its limit, you can check the state of its I/O queue. If this queue is constantly full, this means that the disk is at its peak performance capacity. To investigate the state of the I/O queue for all disks, use the following command:

```
# iostat -x 10
```

The command output will be similar to this:

```
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           4.08    0.00    1.11    0.03    0.03   94.76

Device:  rrqm/s  wrqm/s   r/s   w/s  rkB/s  wkB/s ... %util
sda        0.00    8.30  0.00  3.20   0.00  46.40 ...  0.20
sdb        0.00    0.00  0.00  0.00   0.00   0.00 ...  0.00
sdc        0.00    0.00  0.00  0.00   0.00   0.00 ...  0.00
scd0       0.00    0.00  0.00  0.00   0.00   0.00 ...  0.00
```

You need to pay attention to the following metrics:

- `%iowait` is the percentage of time the CPU was idle while requests were in the I/O queue. If this value is well above zero, this might mean that the node I/O is constrained by the disks speed.
- `%idle` is the percentage of time the CPU was idle while there were no requests in the I/O queue. If this value is close to zero, this means that the node I/O is constrained by the CPU speed.
- `%util` is the percentage of time the device I/O queue was not empty. If this value is close to 100%, this means that the disk throughput is reaching its limit.

iSCSI LUNs performance

iSCSI LUNs served via Virtuozzo Hybrid Infrastructure may experience reduced performance, especially when accessing data with a high number of threads. In this case, it is generally preferable to split the load across multiple smaller iSCSI LUNs, or if possible, avoid iSCSI by accessing storage devices directly. If it is not possible to reduce the size of LUNs, you can consider deploying an LVM RAID0 group over multiple smaller LUNs.

Journal size and location

If your cluster was deployed from an old product version, it is possible that the journal configuration is not optimal. Specifically, if the journal is configured as “inner cache,” that is, stored on the same physical device as data, the recommended size is 256 MB.

Also, consider moving the journals to a faster device such as an SSD or NVMe, if it is applicable to your workload. For more details on storage cache configuration, refer to the Administrator Guide.

Make sure the journal settings are the same for all journals in the same tier.

To check the current size and location of journals, run the following command and specify the desired disk:

```
# vinfra node disk show <DISK>
```

For example:

```
# vinfra node disk show sdc
```

The command output will be similar to this:

```
| service_params | journal_data_size: 270532608
|                | journal_disk_id: 5EE99654-4D5E-4E00-8AF6-7E83244C5E6B
|                | journal_path: /vstorage/94aebe68/journal/journal-cs-b8efe751-
96a6ff460b80
|                | journal_type: inner_cache
```

The journal size and location will be reported as highlighted in this example.

RAM and swap usage

Though the system may operate at nearly 100 percent RAM consumption, it should not aggressively swap virtual memory to and from a disk.

You can check the state of the swap space by running:

```
# free -hm
              total        used        free      shared  buff/cache   available
Mem:           7,7G        2,2G        498M        3,7G        5,0G        1,5G
Swap:          3,9G        352M        3,6G
```

In the command output, the `Swap` row shows the current swap space size. If the reported size is zero, it means swapping is disabled, which may be done intentionally in some configurations.

You can also check the use of swap space by running:

```
# vmstat 1
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu------
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 3  1 244208  10312   1552  62636    4   23    98   249   44  304 28  3 68  1  0
 0  2 244920   6852   1844  67284    0  544  5248   544  236 1655  4  6  0 90  0
 1  2 256556   7468   1892  69356    0 3404  6048  3448  290 2604  5 12  0 83  0
 0  2 263832   8416   1952  71028    0 3788  2792  3788  140 2926 12 14  0 74  0
 0  3 274492   7704   1964  73064    0 4444  2812  5840  295 4201  8 22  0 69  0
```

In the command output, the `si` and `so` columns show the amount of memory swapped from and to a disk, in KB/s. Swap usage may be considered acceptable as long as it well below the device maximum throughput, that is, it does not interfere with the device performance.

S3 service performance

S3 load balancing

We recommend using load balancing at all times. The only scenario that does not benefit from load balancing is when there is a single client. For recommendations on setting up load balancing, refer to the Administrator Guide.

S3 gateways

By default, the S3 service runs with one S3 gateway per node. However, the overall performance can benefit from using multiple gateways per node. The following signs indicate that the number of S3 gateways should be increased:

- The CPU usage of the S3 gateway is near 100 percent.
- The latency of the S3 service is very high (for example, the average latency of more than two seconds).

Note
Changing the number of S3 gateways does not persist across cluster updates. After updating the cluster, you need to apply this change again.

To add an S3 gateway

1. Find out the ID of the object storage volume:

   ```
   # ostor-ctl get-config | grep OBJ
   <...>
   VOL_ID TYPE STATE
   0100000000000002 OBJ READY
   <...>
   ```

2. Check the existing gateways:

   ```
   # ostor-ctl get-config | grep S3GW | grep <vol_id>
   ```

   For example:

   ```
   # ostor-ctl get-config | grep S3GW | grep 0100000000000002
   8000000000000063 0100000000000002 S3GW READY
   svc://0171b0278bcd4534/?address=127.0.0.1:9000
   8000000000000064 0100000000000002 S3GW READY
   svc://56f367af647c9277/?address=127.0.0.1:9000
   ```

```
8000000000000065 0100000000000002 S3GW READY
svc://3d3bbd94a72c1995/?address=127.0.0.1:9000
```

3. Check the mapping of the service URLs to the node IDs:

```
# ostor-ctl get-config -H
HOST_ID HOSTNAME URI ROLES
0171b0278bcd4534 192.168.29.141:2530 vstorage://cluster1/vols/ostor OBJ,FS
56f367af647c9277 192.168.29.157:2530 vstorage://cluster1/vols/ostor OBJ,FS
3d3bbd94a72c1995 192.168.29.22:2530 vstorage://cluster1/vols/ostor OBJ,FS
```

4. Add a new gateway. The operation adds a gateway on the current node, and the port it will use must not be occupied. So, you need to ensure that you are running the command on the relevant node and the port you are specifying is not used. Then, run this command:

```
# ostor-ctl add-s3gw -V <vol_id> -a <IP:port>
```

   For example:

```
# ostor-ctl add-s3gw -V 0100000000000002 -a 127.0.0.1:9001
```

5. Edit the `nginx` configuration file at `/etc/nginx/conf.d/s3-gateway-<VOL_ID>.conf`, and then add the new address in the upstream section at the beginning of the file:

```
upstream s3 {
 server 127.0.0.1:9000;
 server 127.0.0.1:9001;
}
```

6. Reload the `nginx` service:

```
# systemctl reload nginx
```

To remove an S3 gateway

1. Find out the service ID and address of the gateway:

```
# ostor-ctl get-config | grep S3GW
<...>
SVC_ID VOL_ID TYPE STATE URI
8000000000000066 0100000000000004 S3GW READY
svc://0171b0278bcd4534/?address=127.0.0.1:9001
<...>
```

2. Remove the unnecessary gateway from the `nginx` configuration by editing `/etc/nginx/conf.d/s3-gateway-<VOL_ID>.conf` on the relevant node, and then removing the address from the list of upstreams:

```
upstream s3 {
 server 127.0.0.1:9000;
 server 127.0.0.1:9001; # <--- Remove this line
}
```

3. Reload the `nginx` service:

```
# systemctl reload nginx
```

4. Remove the gateway by specifying its service ID:

```
# ostor-ctl rm-s3gw -i <SVC_ID>
```

   For example:

```
# ostor-ctl rm-s3gw -i 8000000000000066
```

When removing a gateway, keep in mind that all ongoing connections to this gateway will be dropped and all ongoing operations may result in a connection error on the client side.

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.
- When using erasure coding, throughput is usually limited at 500-800 MB/s per node, for both reading and writing (depending on specific hardware and workload). If you reach this limit, you may improve your results by spreading the load across a larger number of nodes or by adding more nodes.