Virtuozzo 6.0

Virtuozzo Storage I/O Benchmarking Guide

April 05, 2016

Copyright © 1999-2016 Parallels IP Holdings GmbH and its affiliates. All rights reserved.

Parallels IP Holdings GmbH
Vordergasse 59
8200 Schaffhausen
Switzerland
Tel: + 41 52 632 0411
Fax: + 41 52 672 2010
www.virtuozzo.com
Copyright © 1999-2016 Parallels IP Holdings GmbH and its affiliates. All rights reserved.
This product is protected by United States and international copyright laws. The product's underlying technology, patents, and trademarks are listed at https://virtuozzo.com/wp-content/uploads/2016/02/virtuozzo_legal_notices_20160215.pdf.
Microsoft, Windows, Windows Server, Windows NT, Windows Vista, and MS-DOS are registered trademarks of Microsoft Corporation.
Apple, Mac, the Mac logo, Mac OS, iPad, iPhone, iPod touch, FaceTime HD camera and iSight are trademarks of Apple Inc., registered in the US and other countries.
Linux is a registered trademark of Linus Torvalds.
All other marks and names mentioned herein may be trademarks of their respective owners.

Contents

Introduction

Today's open-source I/O benchmarking tools often utilize incorrect performance testing methodology. The most popular flaws are:

- Use of highly compressible patterns (no data randomness).

  Conducting tests with the standard `dd` tool, many people use `/dev/zero` as the input device. It is well known, however, that some HDDs/SSDs and virtual machines apply special low-level processing to zero data to optimize zero writes. As a result, the write speeds of zero data can differ much from that of real data (which is closer to random).

  > **Note:** Using random input sources like `/dev/urandom` and `/dev/random` is also not an option as these virtual devices offer limited throughput to provide better randomness.

- Measuring of cache performance instead of disk performance.

  Modern systems use multi-level caching to improve overall performance. HDDs have onboard cache (for example, 32MB), RAID controllers may also have cache (for example, 1GB), operating systems can use host memory as disk subsystem cache (which can be tens and hundreds of gigabytes). All of these can have a significant effect on the entire system, so it is important to measure the real performance of disks, not local caches. For this reason, we recommend conducting tests with workloads larger than system caches (for example, larger than the onboard memory) and using long test runs—from 30 to 300 seconds each. It is also important to remember that if write tests do not flush the data, they actually benchmark RAM (OS disk cache), not real disk I/O. To avoid this, make sure your tests use the `sync` operation.

- Use of patterns that require much CPU resources to generate.

  If you use `/dev/urandom` (or, even worse, `/dev/random`), then most of the test run will be spent on creating the workload pattern. Sources like these can be very CPU-intensive and thus are not recommended for I/O benchmarking. Another important aspect is the block size for each I/O operation. For sequential I/O, we recommend to use large blocks (for example, 16MB), because they allow to reach full bandwidth. For random I/O, we recommend to use 4K blocks to benchmark the maximum IOPS limit. Blocks smaller than 4K are useless because such workload scenarios actually test read-modify-write operations instead of random writes. Besides, incorrect hardware configurations also suffer from unaligned I/O operations (see **Improving High-Capacity HDD Performance** in the *Virtuozzo Storage Administrator's Guide*).

- Comparison of apples to oranges.

  When comparing system configurations, choose those with similar redundancy levels. For example, it is incorrect to compare the performance of a cluster with 2 or 3 replicas per data chunk with that of a standalone server without any data redundancy (for example, RAID0). The correct way is to compare a RAID1/5/10 system with a system running Virtuozzo Storage configured to use 2 replicas.

  It also makes sense to measure system performance in real-world scenarios. For example, if you have a cluster of 10 nodes and plan to use each of those for storage and virtualization, run the benchmarking tool on each node in parallel. This real-world scenario will demonstrate actual cluster scalability rather than how fast a a single client can perform.

To ensure benchmarking is done correctly and offer more flexibility to the tester, we created a custom I/O benchmarking tool `at_io_iops`.

Preparations for Benchmarking

To be able to test your Virtuozzo Storage cluster by loading it from a single Node, make the following preparations:

1. Download the test suite available at http://download.openvz.org/~akirov/at_io_iops/ to a local directory you will run the benchmark from. The suite includes:
   - `at_io_iops` — I/O benchmarking tool,
   - `at_io_parallel.sh` — load parallelization tool.

   Notes:

   1. Do not store the test scripts in Virtuozzo Storage.

   2. To run the test suite in a Container or virtual machine, store it in any directory inside the chosen virtual environment.
2. Make both scripts executable with `chmod`.
3. Make a directory for test files according to your purposes: on a local Linux host, on a Virtuozzo Node, on Virtuozzo Storage, in a Container or virtual machine.
4. Configure the test parameters in `at_io_parallel.sh` according to your needs:

Parameter	Description
TIME	Run time in seconds (equals `at_io_iops -t`). Recommended values are 30 and 60. Default: `60`.
FILE_SIZE	Workload file size (equals `at_io_iops -u`). Default: `16G`.
THREADS	The number(s) of threads (equals `at_io_iops -p`). Separate multiple numbers with spaces. Default: `1 4 16`.
ITERATIONS	The number of iterations of the same test, the results of which will be used to calculate the average value. Default: `3`.
REMOTE_BINARIES_PATH	The path on the benchmarked host where `at_io_iops` will be stored. Default: `/root`.
REMOTE_PCS_MOUNT_PATH	The path on the benchmarked host where `at_io_iops` will store workload files. For example: `/pstorage/<cluster_name>`.
AT_IO_LOCAL_PATH	The local path on the client where `at_io_iops` is stored. Default: `/root/perf_test/at_io_iops`.
BLOCKSIZE	The block size for random load (similar to `at_io_iops -s`). Recommended and default: `4K`.

Note: For additional information on `at_io_iops` parameters, see **Benchmark Options** (p. 8).

To be able to emulate real-life scenarios by loading multiple Hardware Nodes of your Virtuozzo Storage cluster at once, make the following additional preparations:

1. Install the `pssh` tool available at https://code.google.com/p/parallel-ssh to the directory with the test suite. For example, if the test suite is stored in the default directory:

```
# cd /root/perf_test/at_io_iops
# wget https://parallel-ssh.googlecode.com/files/pssh-2.3.1.tar.gz
# tar -xvf pssh-2.3.1.tar.gz
# cd pssh-2.3.1
# python setup.py install
```

2. Configure automatic SSH key authentication so that `at_io_parallel.sh` can access benchmarked hosts:

    1. Generate a key on the client with the standard `ssh-keygen` command. For example:

```
# ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
4a:99:03:0c:74:4a:99:03:0c:00:74:87:de:ds:76 root@pstorage.com
The key's randomart image is:
```

    2. Install the generated key on each Node (or Container or virtual machine) you will benchmark. For example:

```
# ssh-copy-id -i ~/.ssh/id_rsa.pub root@10.0.0.1
root@pcs1's password:
Now try logging into the machine, with "ssh 'root@10.0.0.1'", and check in:
  .ssh/authorized_keys
to make sure we haven't added extra keys that you weren't expecting.
# ssh-copy-id -i ~/.ssh/id_rsa.pub root@10.0.0.2
...
```

Recommendations on Benchmarking

The test methodology is vital for performance benchmarking. To be able to obtain repeatable and reliable results, follow these rules:

- Conduct tests with multiple threads: 1, 4, and 16. In real life, many applications are competing for disk resources and loading multiple threads asynchronously. Limiting the benchmark by just one thread (without 4 and 16) would not let you understand the scalability of your system's performance.
- Conduct 30 or 60-second tests. Results of short tests may be strongly affected by caches.
- Run each test at least 3 times and use the average result to exclude scattering.
- Insert 30 to 60-second pauses between test runs. To make sure that the previous test run does not affect the next, introduce a delay between runs so the system has time to flush caches and return to the original not loaded, fully idle state.

- Conduct tests with equal redundancy. Creating additional replicas reduces system performance so make sure that you compare results of tests which you have run with equal redundancy. For example, RAID1 vs. Virtuozzo Storage configured to use 2 replicas.
- Correctly set up Virtuozzo Storage to avoid performance degradation. The following issues degrade Virtuozzo Storage performance:
  - Use of just one Gigabit Ethernet network card which would limit sequential load.
  - Creating RAID volumes for Virtuozzo Storage instead of configuring each disk as a chunk server.
  - Running a chunk server and a metadata server on the same HDD (not applicable to SSDs).
  - Setting up more than 5 metadata servers per cluster.
  - Using too few disks in the cluster (the more disks, the better performance).

Running the Benchmark

When you run `at_io_iops` for the first time, it does the following before benchmarking:

1. Creates files for read or write workload in the specified directory. Each work thread uses its own file.
2. Generates a buffer in memory filled with random data for write workload. The amount of RAM required for the buffer depends on the workload size. The recommendation is to have at least 2GB free RAM.

Note: Creating test workload files may take a significant time.

Next time you run the script, it either uses the existing test workload files if they correspond to the specified options or adjusts these files according to the specified options. After the test workload files have been created, the tool proceeds to benchmark the Virtuozzo Storage cluster for the specified time.

The following typical commands are recommended for benchmarking your cluster with different workload types:

- Sequential read performance, 4 threads:

```
# at_io_iops --read --seq -s 16M --mbs -p 4 -t 30 -u 16G -S --cached -f
/pstorage/<cluster_name>/<benchmark_dir>
```

- Sequential write performance, 4 threads:

```
# at_io_iops --write --seq -s 16M --mbs -p 4 -t 30 -u 16G -S --cached -f
/pstorage/mycluster/benchmark_dir
```

- Random read performance, 16 threads:

```
# at_io_iops --read --rand -s 4K --iops -p 16 -t 30 -u 16G -S --cached -f
/pstorage/mycluster/benchmark_dir
```

- Random write performance, 16 threads:

```
# at_io_iops --fdatasync --rand -s 4K --iops -p 16 -t 30 -u 16G -S --cached -f
/pstorage/mycluster/benchmark_dir
```

- Random writes simulating database performance (data update followed by a transaction commit), 16 threads. 32 random writes are followed with a sync (flush).

```
# at_io_iops --fsync --rand -s 4K -q 32 --iops -p 16 -t 30 -u 16G -S --cached -f
/pstorage/mycluster/benchmark_dir
```

Benchmark Options

The options you can use with `at_io_iops` to benchmark Virtuozzo Storage are described in the following table:

Option	Description
--write|**--read** | **--fsync** | **--fdatasync**	Create a workload: • **--read**: Read workload by means of the standard `read()` system call. • **--write**: Write workload by means of the standard `write()` system call. • **--fsync**: Write workload and call `fsync()` after each `write()`. • **--fdatasync**: Write workload and call `fdatasync()` after each `write()`.
--seq --mbs	Create a sequential workload and report results in megabytes per second.
--rand --iops	Create a random workload and report results as the number of I/O operations per second.
-s <*size*>[**G**|**M**|**K**]	Set block size for a single read or write operation. By varying this parameter you can test cluster performance with different workloads. The recommendation is to use: • `16M` for sequential workloads, • `4K` for random workloads. Using buffers smaller than 4KB would be meaningless as on some configurations the OS would not generate smaller I/O requests correctly because of read-modify-write operations.
-S	Do not wait for `sync()` at the end of the test to exclude the time spent on closing the test files from test results.
-f <*path*>	Set the directory to create test workload files in.
-p <*num*>	Set the number of load threads (and also the number of test files).
-n <*count*>	Use the specified number of files for all threads.
-t <*sec*>	Set the test time in seconds.
-u <*size*>[**G**|**M**|**K**]	Set the total size of all test workload files to be used. For example, if you set `at_io_iops` to create 4 parallel threads and use the option `-u 16G`, four 4GB files will be created. Use 16GB or larger test workload file sets to make the buffer larger than possible caches, including the write-back cache.
--uncached|**--cached**	Enable/disable the use of direct I/O. Use `--cached` unless you plan to run databases and similar applications, especially those configured to work in the direct I/O mode.
-a	Optional. Print test settings without generating actual workload.
-v|**-vv**|**-vvv**	Optional. Set verbosity level if you need all system calls printed.

Benchmarking Multiple Nodes in Parallel

To see how your cluster will do in real life, use `at_io_parallel.sh`, the load parallelization script which does the following:

- Runs `at_io_iops` on multiple hosts in parallel to emulate real-life cluster load.
- Automates execution of multiple sets of tests with different parameters.
- Inserts required pauses between test runs.
- Calculates average results for each set of tests.

Note: The script can run `at_io_iops` on both physical hosts and virtual machines or Containers.

To benchmark the Virtuozzo Storage cluster by loading multiple Nodes at once, run `at_io_parallel.sh` with a list of IP addresses or hostnames to benchmark as an option. When run, `at_io_parallel.sh` copies `at_io_iops` to the specified Nodes and starts it simultaneously on each Node by means of SSH. Before actual benchmarking starts, a warmup is performed to let the workload files be prepared and synchronize the start of `at_io_iops` on all Nodes.

For example, to benchmark three Nodes with the IP addresses 10.0.0.1, 10.0.0.2, 10.0.0.3 and have results stored in `results.txt`, run:

```
# at_io_parallel.sh 10.0.0.1 10.0.0.2 10.0.0.3 | tee results.txt
(C) 2016. Parallels IP Holdings GmbH. All rights reserved.
Preparation...
[1] 17:57:21 [SUCCESS] 10.0.0.1
[2] 17:57:21 [SUCCESS] 10.0.0.2
[3] 17:57:21 [SUCCESS] 10.0.0.3
[1] 17:57:21 [SUCCESS] 10.0.0.1
[2] 17:57:21 [SUCCESS] 10.0.0.2
[3] 17:57:21 [SUCCESS] 10.0.0.3
Loading in 1 threads...
[1] 17:57:21 [SUCCESS] 10.0.0.1
[2] 17:57:21 [SUCCESS] 10.0.0.2
[3] 17:57:21 [SUCCESS] 10.0.0.3
  ---=== Warming (1thr)  ===---
(C) 2004-2016. Parallels IP Holdings GmbH. All rights reserved.
Title:       at_io_iops
Cmdline:     -a --read --seq -s 16M --mbs -p 1 -n 1 -t 1 --cached -u 16G -S
Version:     1.4.0
Processes:   1
Time:        1 sec
Warmup:      0 sec
Testcase:    create/open 1 file(s) of total size 16384.00 MB named (null), start 1
thread(s) and do
             cached sequential read by 16384KB blocks
[1] 17:57:23 [SUCCESS] 10.0.0.1
(C) 2004-2016. Parallels IP Holdings GmbH. All rights reserved.
time: 1.600112 sec; rate: { 99.99300 } MB/s;
[2] 17:57:23 [SUCCESS] 10.0.0.2
(C) 2004-2016. Parallels IP Holdings GmbH. All rights reserved.
time: 1.596777 sec; rate: { 100.20189 } MB/s;
[3] 17:57:23 [SUCCESS] 10.0.0.3
```

```
(C) 2004-2016. Parallels IP Holdings GmbH. All rights reserved.
time: 1.598938 sec; rate: { 100.06642 } MB/s;
    Sum value on iteration 1 = { 300.26125 } MB/s;
---=== Average value = { 300.26125 } MB/s; ===---
  ---=== Sequential read (1thr)  ===---
(C) 2004-2016. Parallels IP Holdings GmbH. All rights reserved.
Title:       at_io_iops
Cmdline:     -a --read --seq -s 16M --mbs -p 1 -n 1 -t 60 --cached -u 16G -S
Version:     1.4.0
Processes:   1
Time:        60 sec
Warmup:      0 sec
Testcase:    create/open 1 file(s) of total size 16384.00 MB named (null), start 1
thread(s) and do cached sequential read by 16384KB blocks
[1] 17:58:33 [SUCCESS] 10.0.0.1
(C) 2004-2016. Parallels IP Holdings GmbH. All rights reserved.
time: 60.069801 sec; rate: { 167.27206 } MB/s;
 [2] 17:58:33 [SUCCESS] 10.0.0.2
(C) 2004-2016. Parallels IP Holdings GmbH. All rights reserved.
time: 60.070686 sec; rate: { 167.26961 } MB/s;
 [3] 17:58:33 [SUCCESS] 10.0.0.3
(C) 2004-2016. Parallels IP Holdings GmbH. All rights reserved.
time: 60.070892 sec; rate: { 167.26903 } MB/s;
    Sum value on iteration 1 = { 501.81070 } MB/s;
 [1] 17:59:33 [SUCCESS] 10.0.0.1
(C) 2004-2016. Parallels IP Holdings GmbH. All rights reserved.
time: 60.009857 sec; rate: { 177.83745 } MB/s;
 [2] 17:59:33 [SUCCESS] 10.0.0.2
(C) 2004-2016. Parallels IP Holdings GmbH. All rights reserved.
time: 60.006870 sec; rate: { 177.84630 } MB/s;
 [3] 17:59:33 [SUCCESS] 10.0.0.3
(C) 2004-2016. Parallels IP Holdings GmbH. All rights reserved.
time: 60.007648 sec; rate: { 177.84399 } MB/s;
    Sum value on iteration 2 = { 533.52774 } MB/s;
 [1] 18:00:34 [SUCCESS] 10.0.0.1
(C) 2004-2016. Parallels IP Holdings GmbH. All rights reserved.
time: 60.061768 sec; rate: { 175.55261 } MB/s;
 [2] 18:00:34 [SUCCESS] 10.0.0.2
(C) 2004-2016. Parallels IP Holdings GmbH. All rights reserved.
time: 60.062653 sec; rate: { 175.55002 } MB/s;
 [3] 18:00:34 [SUCCESS] 10.0.0.3
(C) 2004-2016. Parallels IP Holdings GmbH. All rights reserved.
time: 60.061756 sec; rate: { 175.55264 } MB/s;
    Sum value on iteration 3 = { 526.65527 } MB/s;
---=== Average value = { 520.66457 } MB/s; ===---
```

To get aggregated results of the comparison, you can use `grep` on the results file. For example:

```
# grep '===' results.txt
```

To disable notifications like `Stderr: Warning: Permanently added '10.0.0.1' (RSA) to the list of known hosts.`, run the following command:

```
# echo 'LogLevel=quiet' >> ~/.ssh/config
```

Index