MinIO S3 Throughput Benchmark on NVMe SSD

DECEMBER 2019

MinIO S3 Throughput Benchmark on NVMe SSD

MinIO is a high-performance object storage server designed for AI and ML workloads.

Machine learning, big-data analytics and other AI workloads have traditionally utilized the map-reduce model of computing where data is local to the compute jobs. Modern computing environments have adopted a cloud-native architecture where storage and compute are disaggregated. This enables computing to become stateless, elastic, and scalable independent of storage. Object storage has become the de-facto standard for this architecture.

Applications access data over the network using atomic, immutable object APIs where the data is often in a Binary Large Object (BLOB) format. The relevant performance metrics for object storage are measured in terms of I/O throughput, rather than IOPS.

This document describes the benchmarks that MinIO engineering ran to determine the performance of the MinIO Object Storage Server when run on NVMe. Specifically this document shows how to setup the benchmarking environment, how to run the benchmarking tools and reviews the performance results in detail.

Our results running on a 32 node MinIO cluster can be summarized as follows:

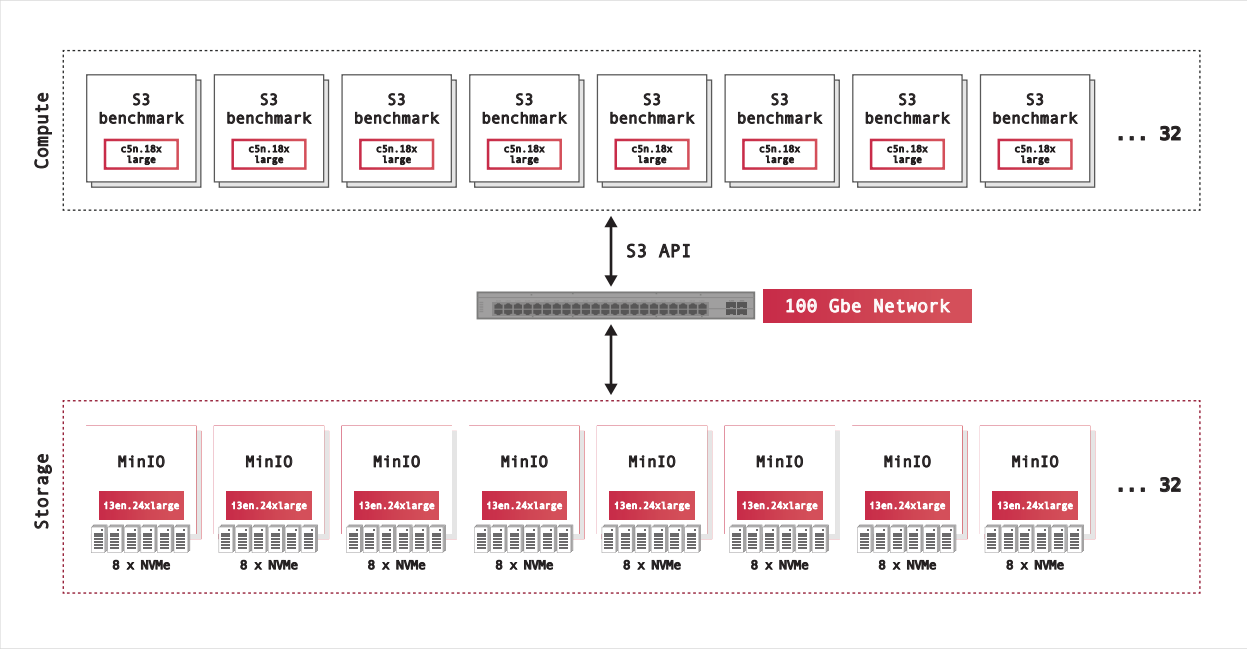
Setup	Avg Read Throughput (GET)	Avg Write Throughput (PUT)
Distributed	183.2 GB/s	171.3 GB/s
Distributed with Encryption	162 GB/s	114.7 GB/s

1. Benchmark Environment

1.1 Hardware

For the purpose of this benchmark, MinIO utilized AWS bare-metal, storage optimized instances with local NVMe drives and 100 GbE networking.

Instance	# Nodes	AWS Instance type	CPU	MEM	Storage	Network
Client	32	c5n.18xlarge	72	192 GB	EBS	100 Gbps
Server	32	i3en.24xlarge	96	768 GB	8 x 7500 GB	100 Gbps



1.2 Software

Property	Value
Server OS	Ubuntu 18.04.2 LTS (Bionic Beaver)
MinIO Version	Minio-RELEASE.2019-05-23T00-29-34Z
Benchmark Tool	S3-benchmark https://github.com/minio/s3-benchmark

1.3 S3-benchmark

MinIO selected the S3-benchmark by wasabi-tech to perform our benchmark tests. This tool conducts benchmark tests from a single client to a single endpoint. During our evaluation, this simple tool produced consistent and reproducible results over multiple runs.

Minor changes were required in s3-benchmark, such as disabling the client-side md5 generation that hindered the tool from saturating the 100 Gbe network.

1.4 Linux Kernel Performance Tuning

Edit the `/etc/sysctl.conf` file to match the following kernel settings:

```
# maximum number of open files/file descriptors
fs.file-max = 4194303

# use as little swap space as possible
vm.swappiness  =  1

# prioritize application RAM against disk/swap cache
vm.vfs_cache_pressure  =  10

# minimum free memory
vm.min_free_kbytes = 1000000

# maximum receive socket buffer (bytes)
net.core.rmem_max = 268435456

# maximum send buffer socket buffer (bytes)
net.core.wmem_max = 268435456

# default receive buffer socket size (bytes)
net.core.rmem_default = 67108864

# default send buffer socket size (bytes)
net.core.wmem_default = 67108864

# maximum number of packets in one poll cycle
net.core.netdev_budget = 1200

# maximum ancillary buffer size per socket
net.core.optmem_max = 134217728

# maximum number of incoming connections
net.core.somaxconn = 65535

# maximum number of packets queued
net.core.netdev_max_backlog = 250000

# maximum read buffer space
net.ipv4.tcp_rmem = 67108864 134217728 268435456

# maximum write buffer space
net.ipv4.tcp_wmem = 67108864 134217728 268435456

# enable low latency mode
net.ipv4.tcp_low_latency = 1
```

```
# socket buffer portion used for TCP window
net.ipv4.tcp_adv_win_scale = 1

# queue length of completely established sockets waiting for accept
net.ipv4.tcp_max_syn_backlog = 30000

# maximum number of sockets in TIME_WAIT state
net.ipv4.tcp_max_tw_buckets = 2000000

# reuse sockets in TIME_WAIT state when safe
net.ipv4.tcp_tw_reuse = 1

# time to wait (seconds) for FIN packet
net.ipv4.tcp_fin_timeout = 5

# disable icmp send redirects
net.ipv4.conf.all.send_redirects = 0

# disable icmp accept redirect
net.ipv4.conf.all.accept_redirects = 0

# drop packets with LSR or SSR
net.ipv4.conf.all.accept_source_route = 0

# MTU discovery, only enable when ICMP blackhole detected
net.ipv4.tcp_mtu_probing = 1
```

Apply these parameters by calling the command `sysctl -p`

The MinIO binary was downloaded onto each server node, and started using the following commands:

```
$ export MINIO_STORAGE_CLASS_STANDARD=EC:2
$ export MINIO_ACCESS_KEY=minio
$ export MINIO_SECRET_KEY=minio123
$ minio server http://minio-{1...32}/mnt/drive{1...32}
```

1.5 Client Setup

Each client was provided with a hostname matching the pattern `client-{1...32}`.

The DNS was configured on the client nodes such that the hostname `minio` on each client resolved to a unique MinIO server. For instance, the URL http://minio:9000 on the `client-1` resolved to the ip address of minio-1, and client-2 resolved to minio-2's ip address etc.

2. Understanding Hardware Performance

2.1 Measuring Single Drive Performance

The performance of each drive was measured using the command dd. DD is a unix tool used to perform bit-by-bit copy of data from one file to another. It provides options to control the block size of each read and write.

Here is a sample of a single NVMe drive's Write Performance with 16MB block-size, O_DIRECT option for a total of 64 copies. Note that we achieve greater than 1.1 GB/sec of write performance for each drive.

```
$ dd if=/dev/zero of=/mnt/drive/test bs=16M count=64 oflag=direct
64+0 records in
64+0 records out
1073741824 bytes (1.1 GB, 1.0 GiB) copied, 0.935502 s, 1.1 GB/s
```

Here is the output of a single HDD drive's Read Performance with 16MB block-size using the O_DIRECT option and a total count of 64. Note that we achieved greater than 2.2 GB/sec of read performance for each drive.

```
$ dd of=/dev/null if=/mnt/drive/test bs=16M count=64 iflag=direct
64+0 records in
64+0 records out
1073741824 bytes (1.1 GB, 1.0 GiB) copied, 0.483597 s, 2.2 GB/s
```

2.2 Measuring JBOD Performance

JBOD performance with O_DIRECT was measured using iozone. Iozone is a filesystem benchmark tool that generates and measures filesystem performance for read, and write among other operations. Iozone command operating with 64 parallel threads, 4MB block-size and O_DIRECT option.

```
$ iozone -t 64 -I -r 4M -s 256M -F /mnt/drive{1..8}/tmp{1..8}
```

23.98 GB/sec of read throughput and 12.939 GB/sec of write throughput on a single node was measured. This represents the throughput combining all drives.

2.3 Network Performance

The network hardware on these nodes allows a maximum of 100 Gbit/sec. 100 Gbit/sec equates to 12.5 Gbyte/sec (1 Gbyte = 8 Gbit).

Therefore, the maximum throughput that can be expected from each of these nodes would be 12.5 Gbyte/sec.

3. Running the 32-node Distributed MinIO benchmark

Run s3-benchmark in parallel on all clients and aggregate results:

```
$ parallel-ssh -O "StrictHostKeyChecking=no" --timeout=0
--hosts=hosts.clients -i "./s3-benchmark -a minio -s minio123 -u
http://minio:9000 -z 10G -t 32 -b s3bench-`hostname` -d 1"
```

The parameters passed into S3-benchmark are as follows:

-a access key
-s secret key
-u endpoint
-z object size
-t number of parallel workers
-b bucket name
-d duration

3.1 Results

The throughput of each of the servers backed by NVMe drives, as measured from each of the clients using s3-benchmark is presented below:

Node #	GET (GB/sec)	PUT (GB/sec)
1	7.3	5.1
2	8.7	4.6
3	4.5	4.9
4	4.5	5
5	4.6	4.9
6	4.5	4.8
7	4.9	4.9
8	4.8	4.9
9	5.2	4.7
10	4.4	4.9
11	5.2	4.9
12	5.2	4.9
13	5.2	4.5
14	5.2	4.9
15	5.3	4.9
16	5.3	4.5
17	5.7	5.1
18	5.7	4.6
19	5.6	7.7
20	5.1	4.7
21	5.8	7.4
22	5.9	4.7
23	6	4.9
24	6.1	7.4
25	6.2	7.1
26	6.3	7.7
27	6.7	4.7
28	6.2	4.7
29	6.4	5.6
30	6.6	5.4
31	7.2	5.8
32	7	6.5
Aggregate	**183.2**	**171.3**

3.2 Interpretation of Results

The average network bandwidth utilization during the write phase was 77 Gbit/sec and during the read phase was 84.6 Gbit/sec. This represents client traffic as well as internode traffic. The portion of this bandwidth available to clients is about half for both reads and writes.

The network was almost entirely choked during these tests. Higher throughput can be expected if a dedicated network was available for internode traffic.

Note that the write benchmark is slower than read because benchmark tools do not account for write amplification (traffic from parity data generated during writes). In this case, the 100 Gbit network is the bottleneck as MinIO gets close to hardware performance for both reads and writes.

3.3 Impact of encryption

The results of the same tests with encryption enabled are presented below:

Node #	GET (GB/sec)	PUT (GB/sec)
1	5.1	3.6
2	5.1	3.6
3	5	3.6
4	4.9	3.7
5	5.1	3.7
6	5	3.5
7	5	3.7
8	5.2	3.6
9	5.2	3.6
10	5.1	3.5
11	5.1	3.5
12	5.1	3.5
13	5.2	3.5
14	5.1	3.6
15	5.1	3.6
16	5	3.5
17	5	3.6
18	5	3.6
19	5.2	3.5
20	4.9	3.6
21	5.1	3.4
22	5.1	3.5
23	5.1	3.5
24	5.1	3.6

25	5.1	3.6
26	5.1	3.6
27	5	3.7
28	5.1	3.7
29	5	3.6
30	5	3.6
31	5	3.7
32	4.9	3.6
Aggregate	**61.9**	**114.7**

The impact of encryption is negligible for reads. This reduction can be accounted for by the overhead of decrypting the objects while reading. In the case of writes, the overhead of encryption at a large scale impacted the performance. In both the cases, the CPU was the bottleneck to performance. However, the overall speed is still very high in comparison to the maximum available bandwidth, therefore we strongly recommend turning on encryption for all production setups.

4. Conclusion

Based on the results above, we found that MinIO takes complete advantage of the available hardware. Its performance is only constrained by the underlying hardware available to it. This has been tested with the most powerful hardware available on AWS.