ORACLE

Running Benchmarks on OCI File Storage with Lustre

August 2025, Version 1.0

Public

Disclaimer

This document in any form, software or printed matter, contains proprietary information that is the exclusive property of Oracle. Your access to and use of this confidential material is subject to the terms and conditions of your Oracle software license and service agreement, which has been executed and with which you agree to comply. This document and information contained herein may not be disclosed, copied, reproduced or distributed to anyone outside Oracle without prior written consent of Oracle. This document is not part of your license agreement nor can it be incorporated into any contractual agreement with Oracle or its subsidiaries or affiliates.

This document is for informational purposes only and is intended solely to assist you in planning for the implementation and upgrade of the product features described. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, timing, and pricing of any features or functionality described in this document remains at the sole discretion of Oracle. Due to the nature of the product architecture, it may not be possible to safely include all features described in this document without risking significant destabilization of the code.

Acknowledgements

- Aboo Valappil and Matt Hambrick, OCI Storage
- Jon Shelley, OCI Solutions Architect
- Sudhir Upreti, OCI Solutions Architect
- Heinz Mielimonka, OCI Storage Specialist

ORACLE

Table of Contents

Overview

This document details the benchmark methodologies and results for Oracle Cloud Infrastructure (OCI) File Storage with Lustre, a fully managed, high-performance, parallel filesystem designed for AI, ML, and high-performance computing (HPC) workloads. It is intended for architects, engineers, and IT decisionmakers who need clear, reproducible evidence of Lustre's capability to deliver scalable, predictable, and robust file storage performance in OCI.

This document has technical guidance for running standardized benchmarks to evaluate throughput, IOPS, and metadata operations in real-world, large-scale environments. The benchmarks use tools such as IOR (Interleaved or Random), FIO (Flexible I/O Tester), and MDTest to simulate diverse I/O patterns and measure performance across varying system sizes and configurations.

Running IOR

IOR is a parallel I/O benchmarking tool designed to test the data throughput (read/write bandwidth) of HPC filesystems by using message passing interface (MPI). IOR supports different access patterns—for example, POSIX versus MPI-IO API, single file versus file per process, sequential versus random, and collective versus independent—and can run across many nodes and threads to simulate real-world HPC workloads.

The IOR benchmark is available from its official repository at github.com/hpc/ior.

IOR can be run either directly from the command line or through the use of a configuration file. For File Storage with Lustre, the tests were performed using an IOR configuration file, with the command run as follows:

```
/opt/openmpi-4.1.4/bin/mpirun --mca pml ob1 --mca btl tcp,self --mca btl_tcp_if_include eth0 --
machinefile ${num_clients}node -npernode ${num_threads} /mnt/lustrefs/sudhir/ior/ior-
4.0.0/src/ior -f ${ior_file}
```

Single Thread and Single Compute Node Performance

Performance is limited by the capabilities of a single process or thread and a single node. It doesn't demonstrate the aggregate performance capabilities of the File Storage with Luster filesystem.

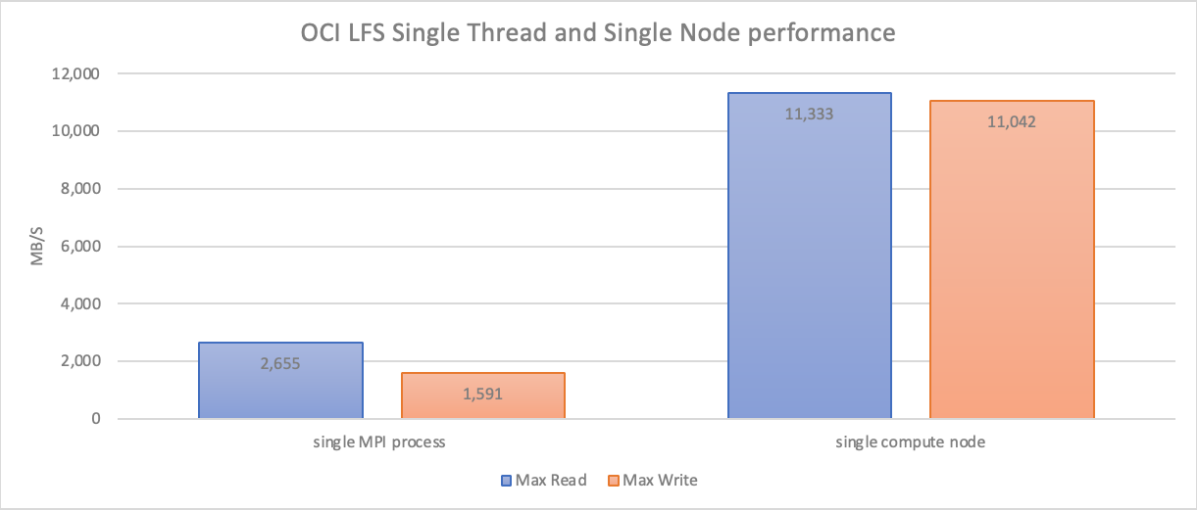

Figure 1: File Storage with Lustre Single Thread and Single Node Performance

Multiple-Node Performance

IOR testing was performed under the following scenarios to evaluate and compare the performance of a 125-TB filesystem and a 250-TB filesystem after it was scaled up. The 125-MB per TB performance tier was used.

- POSIX: File per process or task
- POSIX: Single shared file
- MPI-IO: File per process or task
- MPI-IO: Collective I/O with single shared file (complex workload)

Performance numbers scale linearly across the two configurations tested, 125 TB with 16 clients and 250 TB with 32 clients.

Figure 2: IOR Throughput Comparison of File Storage with Lustre—125 TB (16 Clients) and 250 TB (32 Clients)

For all tests, files were striped across all Object Storage targets (OSTs) in the Lustre filesystem, which is particularly beneficial in single-shared-file scenarios. To avoid caching, the block/data size per client was configured to exceed 1.5 times the available client memory.

```
lfs setstripe -c -1 -S 10M /mnt/lustrefs/${workdir) ##[Stripe count set to -1 to stripe across all OSTs, Stripe Size 10M to match the transfer and block size of the IOR workload]
```

The following sections show the contents of the `ior_file` configuration file for each of the multiple-node performance tests.

POSIX: File per Process or Task

```
##POSIX File per Process
IOR START

### You MUST change the following parameters
    ### numTasks=1              # number of MPI processes to use.  You may choose to use one or
more MPI processes per node.
    segmentCount=6400       # must be > fileSize / ( blockSize * numTasks ) where fileSize must
be greater than 1.5 times the aggregate DRAM available for use by the page cache
#    memoryPerNode=80%       # must be > 80% of the node's DRAM available for use by the page
cache

### You MAY change the following parameters
    transferSize=10M         # size of a single data buffer to be transferred by a single I/O
call.  You must find the optimal transferSize for the storage system.
    blockSize=10M            # must be the same as transferSize
    testFile=/mnt/lustrefs/sudhir/data/datafile.dat   # will read/write files called
./datafile.dat.00000000 ./datafile.dat.00000001 etc
    collective=0             # for MPIIO api use collectives (e.g. MPI_File_read_all) instead of
independent calls (e.g. MPI_File_read)
    keepFile=0               # do not delete files used by the test at the end of each execution

### You MUST NOT change the following parameters
    reorderTasksConstant=1  # each node n writes data; that data is then read by node n+1
    intraTestBarriers=1     # use barriers between open/read/write/close
    repetitions=1           # executes the same test multiple times
    verbose=2               # print additional information about the job geometry
    fsync=1                 # for POSIX api call fsync(2) before close(2)

### The following parameters define the nature of the benchmark test
    api=POSIX               # use POSIX open/close/read/write
    filePerProc=1           # read/write one file per MPI process
    randomOffset=0          # randomize order in which reads/writes occur
    writeFile=1             # perform the write component of the test
    readFile=0              # perform the read component of the test after the write component
has completed

RUN

IOR STOP
```

POSIX: Single Shared File

```
##POSIX Single Shared File
IOR START

### You MUST change the following parameters
    ### numTasks=1              # number of MPI processes to use.  You may choose to use one or
more MPI processes per node.
    segmentCount=5110       # must be > fileSize / ( blockSize * numTasks ) where fileSize must
be greater than 1.5 times the aggregate DRAM available for use by the page cache
   # memoryPerNode=90%       # must be > 80% of the node's DRAM available for use by the page
cache

### You MAY change the following parameters
    transferSize=10M         # size of a single data buffer to be transferred by a single I/O
call.  You must find the optimal transferSize for the storage system.
    blockSize=10M            # must be the same as transferSize
    testFile=/mnt/lustrefs/sudhir/data/datafile.dat   # will read/write to a shared file called
datafile.dat
    collective=0            # for MPIIO api use collectives (e.g. MPI_File_read_all) instead of
independent calls (e.g. MPI_File_read)
    keepFile=1              # do not delete files used by the test at the end of each execution
### You MUST NOT change the following parameters
    reorderTasksConstant=1  # each node n writes data; that data is then read by node n+1
    intraTestBarriers=1     # use barriers between open/read/write/close
    repetitions=1           # executes the same test multiple times
    verbose=2               # print additional information about the job geometry
    fsync=1                 # for POSIX api call fsync(2) before close(2)

### The following parameters define the nature of the benchmark test
    api=POSIX               # use POSIX open/close/read/write
    filePerProc=0           # read/write one file per MPI process
    randomOffset=0          # randomize order in which reads/writes occur
    writeFile=1             # perform the write component of the test
    readFile=1              # perform the read component of the test after the write component
has completed

RUN

IOR STOP
```

MPI-IO: File per Process or Task

```
##MPIIO Filepertask##
IOR START

### You MUST change the following parameters
    ### numTasks=1              # number of MPI processes to use.  You may choose to use one or
more MPI processes per node.
    segmentCount=327680       # must be > fileSize / ( blockSize * numTasks ) where fileSize
must be greater than 1.5 times the aggregate DRAM available for use by the page cache
#    memoryPerNode=80%        # must be > 80% of the node's DRAM available for use by the page
cache

### You MAY change the following parameters
    transferSize=10M          # size of a single data buffer to be transferred by a single I/O
call.  You must find the optimal transferSize for the storage system.
    blockSize=10M             # must be the same as transferSize
    testFile=/mnt/lustrefs/sudhir/data/datafile.dat   # will read/write files called
./datafile.dat.00000000 ./datafile.dat.00000001 etc
    collective=1              # for MPIIO api use collectives (e.g. MPI_File_read_all) instead of
independent calls (e.g. MPI_File_read)
    keepFile=0                # do not delete files used by the test at the end of each execution

### You MUST NOT change the following parameters
    reorderTasksConstant=1  # each node n writes data; that data is then read by node n+1
    intraTestBarriers=1     # use barriers between open/read/write/close
    repetitions=1           # executes the same test multiple times
    verbose=2               # print additional information about the job geometry
    fsync=1                 # for POSIX api call fsync(2) before close(2)

### The following parameters define the nature of the benchmark test
    api=MPIIO               # use POSIX open/close/read/write
    filePerProc=1           # read/write one file per MPI process
    randomOffset=0          # randomize order in which reads/writes occur
    writeFile=1             # perform the write component of the test
    readFile=1              # perform the read component of the test after the write component
has completed

RUN

IOR STOP
```

MPI-IO: Single Shared File

```
##MPIIO Single Shared File
IOR START

### You MUST change the following parameters
    ### numTasks=8              # number of MPI processes to use.  You may choose to use one or
more MPI processes per node.
    segmentCount=10220      # must be > fileSize / ( blockSize * numTasks ) where fileSize must
be greater than 1.5 times the aggregate DRAM available for use by the page cache
    #memoryPerNode=80%       # must be > 80% of the node's DRAM available for use by the page
cache
    #segmentCount=100
### You MAY change the following parameters
    transferSize=10M         # size of a single data buffer to be transferred by a single I/O
call.  You must find the optimal transferSize for the storage system.
    blockSize=10M            # must be the same as transferSize
    testFile=/mnt/lustrefs/sudhir/data/datafile.dat   # will read/write to a shared file called
datafile.dat
    collective=1            # for MPIIO api use collectives (e.g. MPI_File_read_all) instead of
independent calls (e.g. MPI_File_read)
    keepFile=1              # do not delete files used by the test at the end of each execution

### You MUST NOT change the following parameters
    reorderTasksConstant=1  # each node n writes data; that data is then read by node n+1
    intraTestBarriers=1     # use barriers between open/read/write/close
    repetitions=1           # executes the same test multiple times
    verbose=2               # print additional information about the job geometry
    fsync=1                 # for POSIX api call fsync(2) before close(2)

### The following parameters define the nature of the benchmark test
    api=MPIIO               # use POSIX open/close/read/write
    filePerProc=0           # read/write one file per MPI process
    randomOffset=0          # randomize order in which reads/writes occur
    writeFile=1             # perform the write component of the test
    readFile=1             # perform the read component of the test after the write component
has completed

RUN

IOR STOP
```

Running FIO

FIO is a highly configurable and portable I/O workload generator used to benchmark and simulate different types of block I/O workloads. FIO supports sequential and random reads/writes (including mixed workloads with different percentages of reads and writes), IOPS, latency measurements, and POSIX and libaio APIs (including async and directIO). FIO can be run on raw devices, filesystems, or networked storage.

The FIO benchmark was run for three different scenarios:

- Mixed random reads/writes with a ratio of 60% reads to 40% writes for a 1M block size (buffered I/O)
- Mixed random reads/writes with a ratio of 60% reads to 40% writes for a 128k block size (Direct I/O)
- Mixed random reads/writes with a ratio of 60% reads to 40% writes for a 4k block size (Direct I/O)

For mixed random reads/writes with a 1M block size, buffered I/O was used instead of Direct I/O. The total data size per client was set sufficiently large to avoid caching effects. Performance numbers are scaling almost linearly with OCI File Storage with Lustre capacity.

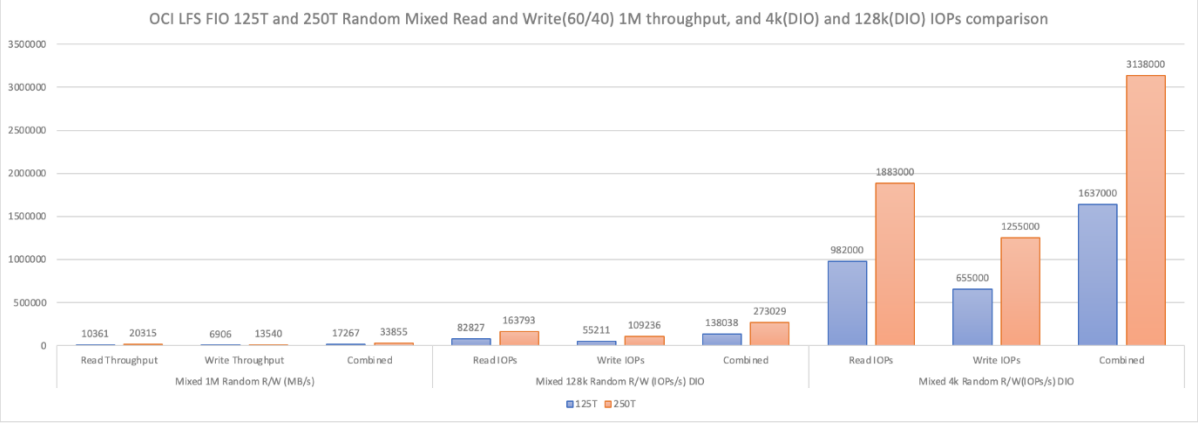

Figure 3: File Storage with Lustre 125 TB and 250 TB Random Mixed Read/Write (60%-40%) 1M Throughput and 4k and 128k (Direct I/O) IOPS Comparison

Following are the contents of the FIO job file:

```
[global]
name=fio-rand-RW
directory=/mnt/lustrefs/fio/data/
rw=randrw
rwmixread=60
rwmixwrite=40
bs=${bs}
direct={1 for direct, 0 for buffered}
numjobs=${num_jobs}
time_based
group_reporting
runtime=300

[file1]
size=2G
ioengine=libaio
iodepth=${iodepth}
```

Following are the FIO commands used for the server/client FIO run, which takes `hostfile` as an argument:

```
HOSTFILE=$1
FIO_JOB=$2

# Start fio servers on all remote hosts
echo "Starting fio servers"
clush -w $(cat $HOSTFILE | awk '{print $1}' | paste -sd,) "sudo nohup /opt/storage-
benchmark/fio/bin/fio --server > /tmp/fio_server.log 2>&1 &"

# Run the test
echo "Running fio client with hostfile..."
/opt/storage-benchmark/fio/bin/fio --client=$HOSTFILE $FIO_JOB

# Stop remote fio servers
clush -w $(cat $HOSTFILE | awk '{print $1}' | paste -sd,) "sudo pkill fio"
```

Metadata Testing Using MDTest

MDTest is a metadata performance benchmark used to measure how well a filesystem handles metadata operations such as file and directory creation, stat, and deletion at scale. MDTest uses MPI to scale across multiple nodes and directories, which makes it ideal for assessing parallel filesystem efficiency (for example, Lustre, GPFS, and BeeGFS).

Performance testing was conducted on a 125-TB filesystem with a single metadata target (MDT) and a 250-TB filesystem with two MDTs to compare and evaluate performance across the two configurations. Tests were performed using both single-directory and unique-directory scenarios. To minimize caching effects, the total number of files was set to over 1 million.

As shown in the following chart, MDTest performance exhibited a linear scalability with an increasing number of MDTs and overall filesystem capacity.

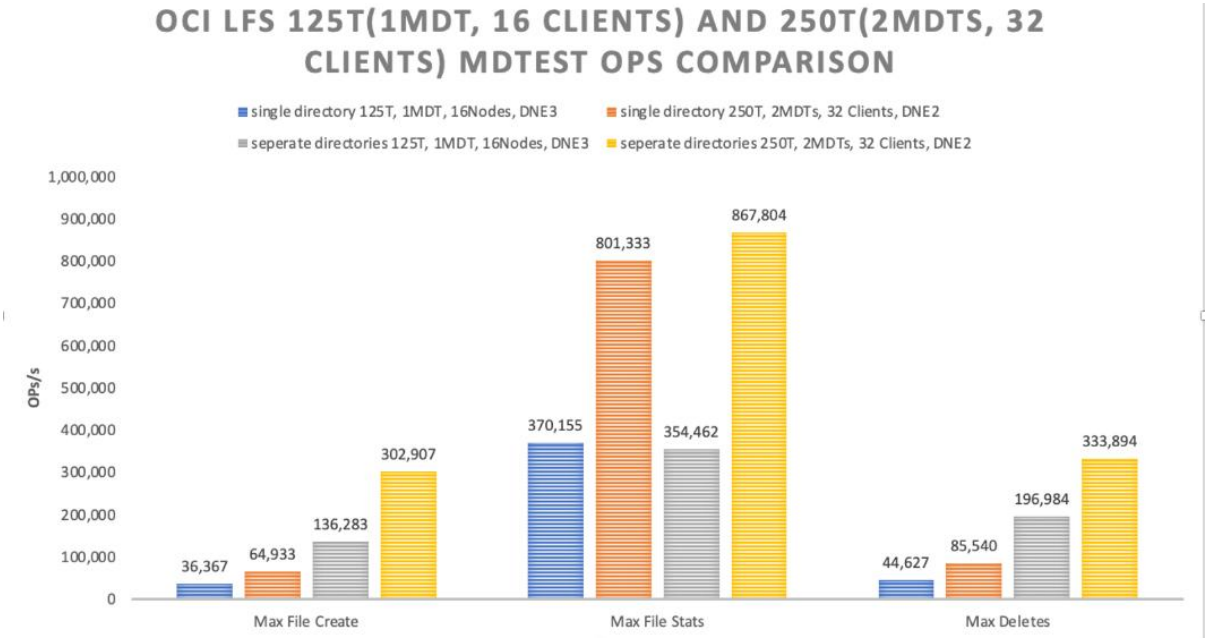

Figure 4: File Storage with Lustre 125-TB (1 MBT, 16 Clients) and 250-TB (2 MBTs, 32 Clients) MDTest Ops Comparison

For testing on the 250-TB filesystem, directories were striped across two MDTs using the following command:

```
lfs setdirstripe –c 2 –D /mnt/lustrefs/${workdir} ##[Utilize both MDTs in the OCI LFS]
```

The MDTest benchmarks were run using the following commands:

```
##Single Directory
/opt/openmpi-4.1.4/bin/mpirun --oversubscribe --mca pml ob1 --mca btl tcp,self --mca
btl_tcp_if_include eth0 --machinefile ${clients}node -npernode ${ppn}
/mnt/lustrefs/sudhir/ior/ior/ior-4.0.0/src/mdtest -F -C -r -T -n ${numfiles} -d
/mnt/lustrefs/mdtest/data2

##Separate/Unique Directories
/opt/openmpi-4.1.4/bin/mpirun --oversubscribe --mca pml ob1 --mca btl tcp,self --mca
btl_tcp_if_include eth0 --machinefile ${clients}node -npernode ${ppn}
/mnt/lustrefs/sudhir/ior/ior/ior-4.0.0/src/mdtest -F -C -r -T -u -n ${num_files} -d
/mnt/lustrefs/mdtest/data2
```

Performance Monitoring While Running Benchmarks

OCI provides monitoring services for OCI File Storage with Lustre. Metrics are available from the **Metrics** link on the details page of the Lustre file system in the Oracle Cloud Console. Metrics are also available from OCI Metrics Explorer, which provides advanced metric queries using Monitoring Query Language (MQL) and the ability to define alarms on available metrics when they breach certain levels.

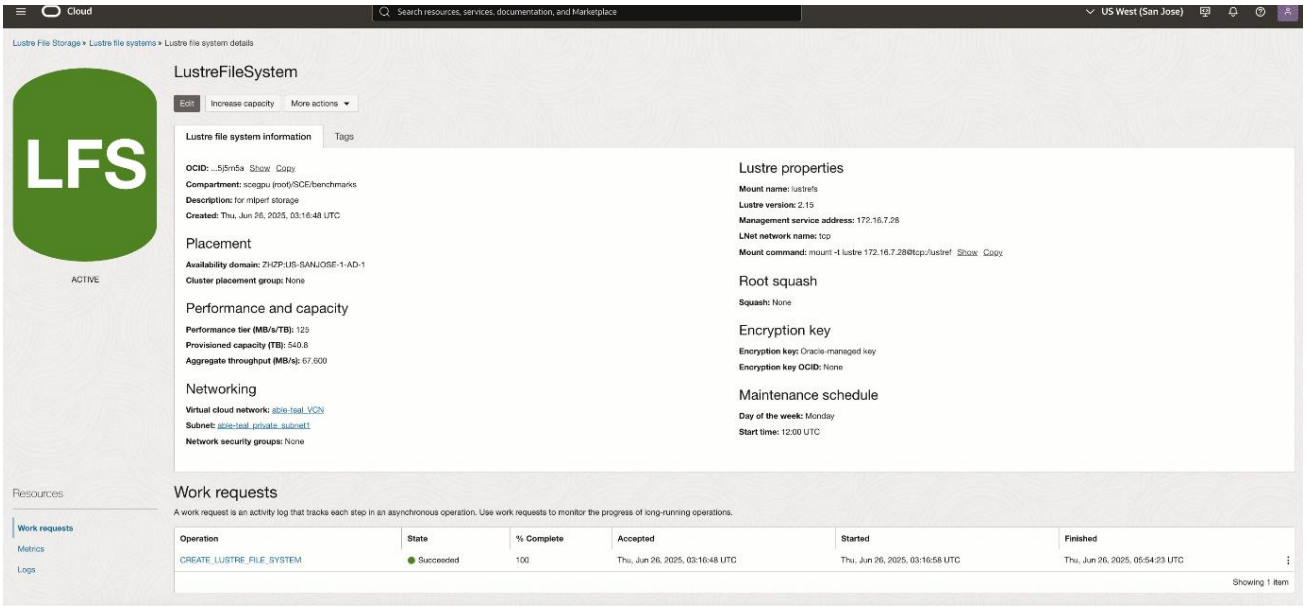

Figure 5: Details Page for a Lustre File System in the Oracle Cloud Console

The following table describes the available metrics.

Metric Type	Metric Name	Description
Capacity	FileSystemCapacity	This metric has two dimensions, *total* and *available*, to provide the total filesystem size and free space.
Inode Capacity	FileSystemInodeCapacity	Similar to FileSystemCapacity, this metric has *total* and *available* dimensions, to show the total inodes in the filesystem and the free inodes in the filesystem.
Bytes Read	ReadThroughput	This metric is bytes read from the filesystem per one-minute intervals. Divide this number by 60 to get the read throughput.
Bytes Written	WriteThroughput	This metric is bytes written to the filesystem per one- minute intervals. Divide this number by 60 to get the write throughput.
Read Operations	DataReadOperations	This metric is the number of read operations from the file system per one-minute intervals. Divide this number by 60 to get IOPS.
Write Operations	DataWriteOperations	This metric is the number of write operations from the file system per one-minute intervals. Divide this number by 60 to get IOPS.
Metadata Operations	MetadataOperations	This metric provides the metadata operations in the filesystem. It's separated into various operations using the *operation* metric dimension. The *operation* can be open, close, mknod, link, unlink, mkdir, rmdir, rename, getattr, setattr, getxattr, setxattr, and statfs.

The following image shows default metric visualizations available in the Oracle Cloud Console. These default metric charts already have aggregations in place.

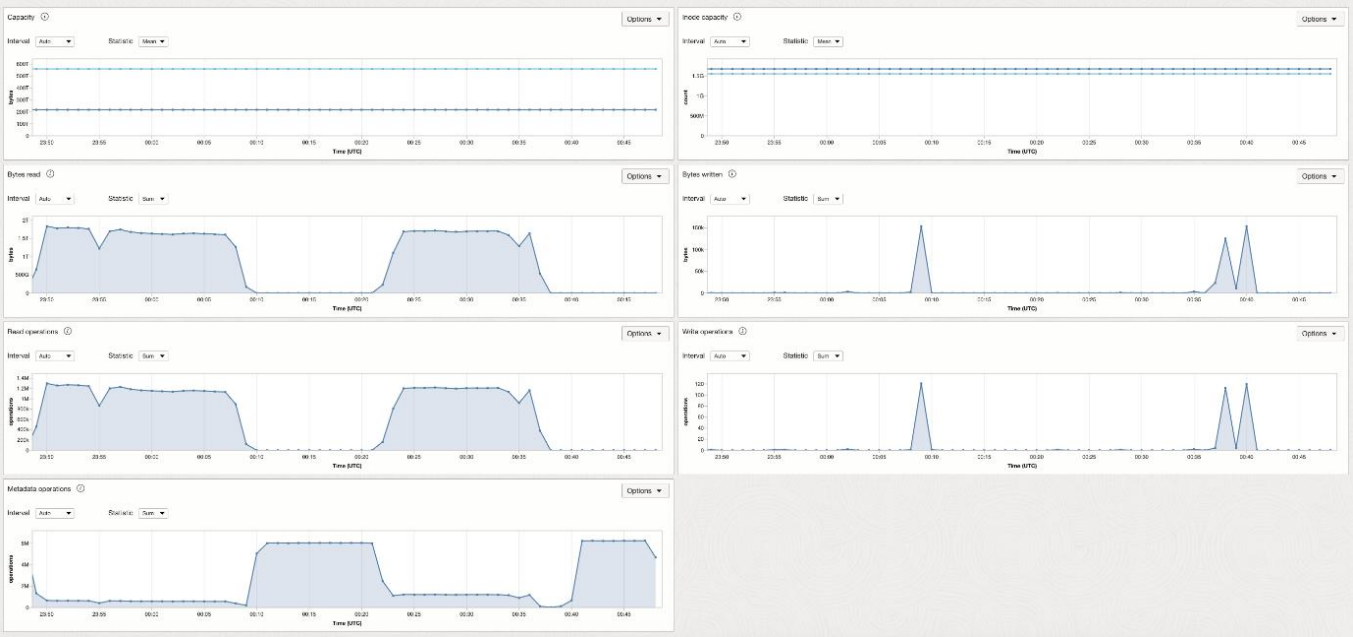

Figure 6: Available Metrics for a Lustre File System in the Oracle Cloud Console

The definition (MQL) for each chart can be accessed from the **Options** menu. For advanced customization and defining alarms on thresholds, use Metrics Explorer. All the preceding metrics are available in the `oci_lustrefilesystem` metric name space.

Conclusion

The benchmarking results in this document confirm that OCI File Storage with Lustre consistently delivers linear and predictable scalability in throughput, IOPS, and metadata operations as filesystem capacity and client count increase. Using industry-standard tools—I/O-workload generators (IOR and FIO) and a metadata tester (MDTest)—demonstrates that performance scales proportionally with storage size, effectively supporting demanding AI, ML, and HPC workloads. The evaluation covers both single-node and multiple-node scenarios, multiple file access patterns, and varied block sizes, demonstrating robust performance. OCI's integrated monitoring further enables users to track metrics and fine-tune performance. These findings reinforce OCI File Storage with Lustre as a reliable and high-performance storage solution for enterprises seeking to accelerate and simplify the management of large-scale, data-intensive workloads.

If you have questions or specific performance requirements, contact us by email or through OCI help and support.

Related Documents

- Deploy a scalable, distributed file system using Lustre (reference architecture)
- Generally Available: Fully Managed Lustre File Storage in the Cloud (blog post)
- Overview of File Storage with Lustre (documentation)

ORACLE

Connect with us

Call +**1.800.ORACLE1** or visit **oracle.com**. Outside North America, find your local office at **oracle.com/contact**.

blogs.oracle.com facebook.com/oracle twitter.com/oracle