All NAS systems share several filesystems intended to provide working space for compute jobs. These filesystems, called "nobackup" (/nobackupp*X*), provide more than 90 petabytes (PB) of disk space and serve many thousands of cores. The /nobackup filesystems are Lustre-based filesystems managed under Lustre software version 2.*x*.

Using the Lustre /nobackup Filesystems

As the name suggests, these "nobackup" filesystems are for temporary use, and are not backed up. Lustre can handle many large files, but you cannot use the Lustre filesystems for long-term storage. If you want to save your files, move them to Lou.

Each Lustre filesystem is shared among many users. To learn how to achieve good I/O performance for your applications and avoid impeding the I/O operations of other users, read the related articles listed at the bottom of the page.

WARNING: Any files that are removed from Lustre /nobackup filesystems *cannot* be restored. You should store essential data on the Lou mass storage system (lfe[*5-8*]) or on other, more permanent storage.

Which Lustre Filesystem is Assigned to Me?

Once you are granted a NAS account, one of the Lustre filesystems will be assigned to you. To find out which one, run the ls command as follows:

```
% ls -l /nobackup/your_username
```

In the output, look for the filesystem name (nobackupp*X*). For example, the following output shows that the user's assigned filesystem is /nobackupp17:

```
lrwxrwxrwx 1 root root 19 Sep 23  2021 /nobackup/your_username -> /nobackupp17/your_username
```

Default Quota and Policy on /nobackup Filesystems

Disk space and inode quotas are enforced on the /nobackup filesystems. The default soft and hard quota limits for inodes are 500,000 and 600,000, respectively. Quotas for the disk space are 1 terabyte and 2 terabytes, respectively. To check your disk space, inode usage, and quota on your /nobackup filesystem, run the lfs command as follows:

```
% lfs quota -h -u username /nobackup/username
Disk quotas for user username (uid nnnn):
     Filesystem    used   quota   limit   grace   files   quota   limit   grace
/nobackup/username
                     4k     1.024T  2.048T      -       1  500000  600000      -
```

It is expected that your data will occasionally exceed the soft limit. However, after a 14-day grace period, your access to the batch queues will be restricted and you will not be able to run new jobs until your data is reduced below the soft limit. If you reach the hard limit, your batch access will immediately be restricted, but any running jobs will continue.

Once you exceed your filesystem limits, PBS will return the following error and prevent you from submitting new jobs:

```
qsub: Filesystem quota exceeded! Please reduce your usage or request an increase before submitting.
```

If you anticipate a long-term need for higher quota limits, please send a justification via email to support@nas.nasa.gov. Your request will be reviewed by HECC management for approval.

For more information, see Quota Policy on Disk Space and Files.

Note: When a Lustre filesystem is full, the jobs writing to it will hang. A Lustre error with code -28 in the system log file indicates that the filesystem is full. NAS support staff will typically send emails to those using the most space, with a request to clean up their files.

Lustre Filesystem Configuration: Progressive File Layout

Lustre's Progressive File Layout (PFL) feature enables filesystems to dynamically change the stripe count for files based on their size. This means that the files are dynamically striped, therefore, you should ***not*** set custom stripe size or stripe counts.

The Lustre filesystems are called /nobackupp[*17-19, 27-29*]. They are equipped with a small number of solid state drives (SSDs) in addition to the hard disk drives (HDDs).

The HDD configurations are shown in the following table with the abbreviation nbp*X*. In the table, P = petabytes; T = terabytes.

Lustre PFL HDD Configurations

Filesystem	nbp17,18	nbp19	nbp27,28	nbp29
# of OSTs	32	16	32	16
size/OST	565 T	565 T	565 T	565 T
Total Space	18.6 P	9.3 P	18.6 P	9.3 P

Different PFL configurations are set among the SSD and HDD pools. To learn more, see Progressive File Layout with SSD and HDD Pools.

A Lustre filesystem is a highly scalable, high-performance shared filesystem for Linux clusters that is managed with Lustre software. It can support many thousands of client nodes, petabytes of storage, and hundreds of gigabytes per second of I/O throughput. The NAS Lustre filesystems are named "/nobackupp*X*."

Each Lustre filesystem is actually a set of many small filesystems, which are referred to as object storage targets (OSTs). The Lustre software presents the OSTs as a single unified filesystem.

For more information about OSTs and other Lustre filesystem components, see the Main Lustre Components section below.

Useful Lustre Commands

The examples in this section use /nobackupp*17* as an example of a specific Lustre filesystem.

Listing Disk Usage and Quotas

To display disk usage and limits on your /nobackup directory:

pfe*21*% lfs quota -h -u *username* /nobackupp*17*

or

pfe*21*% lfs quota -h -u *username* /nobackup/*username*

To display usage on each OST, add the -v option:

pfe*21*% lfs quota -h -v -u *username* /nobackup/*username*

Listing Space Usage

To list space usage per OST and MDT in human-readable format for all Lustre filesystems or for a specific one:

pfe*21*% lfs df -h
pfe*21*% lfs df -h /nobackupp*17*

Listing Inode Usage

To list inode usage for all filesystems or for a specific one:

pfe*21*% lfs df -i
pfe*21*% lfs df -i /nobackupp*17*

Listing OSTs

To list all the OSTs for the filesystem:

pfe*21*% lfs osts /nobackupp*17*

Viewing Striping Information

To view the striping information for a specific file or directory:

pfe*21*% lfs getstripe *filename*
pfe*21*% lfs getstripe -d *directory_name*

Note: Omitting the -d flag will display striping for all files in the directory.

File Striping

Files on the Lustre filesystems are striped automatically. This means they are transparently divided into chunks that are written or read simultaneously across a set of OSTs within the filesystem. The chunks are distributed among the OSTs using a method that ensures load balancing. Files larger than 100 gigabytes (GB) *must* be striped in order to avoid taking up too much space on any single OST, which might adversely affect the filesystem.

Striping benefits include:

- Allows one or more clients to read/write different parts of the same file at the same time, providing higher I/O bandwidth to the file because the bandwidth is aggregated over the multiple OSTs.
- Allows file sizes larger than the size of a single OST.

However, striping small files increases the number of objects in each file, resulting in increased overhead due to network operations —sometimes creating server contention. This is not likely to cause problems for files larger than 4 megabytes (MB), but in many

cases, striping can cause noticeably slower read/write performance for files smaller than 4 MB.

Progressive File Layout

Lustre's progressive file layout (PFL) feature enables stripe settings for different segments of a file to be configured in such a way that a small stripe count is used for the beginning segment, and larger stripe counts are used for latter segments as the file grows. All of the NAS Lustre filesystems have been configured with NAS-selected PFL settings.

To learn more about PFL, see Progressive File Layout with SSD and HDD Pools.

Main Lustre Components

Lustre filesystem components include:

Metadata Server (MDS)
: Service nodes that manage all metadata operations, such as assigning and tracking the names and storage locations of directories and files on the OSTs. There is 1 MDS per filesystem.

Object Storage Target (OST)
: Storage devices where users' file data is stored. The size of each OST varies from approximately 7 terabytes (TB) to approximately 22 TB, depending on the Lustre filesystem. The capacity of a Lustre filesystem is the sum of the sizes of all OSTs. There are multiple OSTs per filesystem.

Metadata Target (MDT)
: A storage device where the metadata (name, ownership, permissions and file type) are stored. There is 1 MDT per filesystem.

Object Storage Server (OSS)
: Service nodes that run the Lustre software stack, provide the actual I/O service and network request handling for the OSTs, and coordinate file locking with the MDS. Each OSS can serve multiple OSTs. The aggregate bandwidth of a Lustre filesystem can approach the sum of bandwidths provided by the OSSes. There can be 1 or multiple OSSes per filesystem.

Lustre Clients
: Compute nodes that mount the Lustre filesystem, and access/use data in the filesystem. There are typically thousands of Lustre clients per filesystem.

Hard disk drives (HDDs) and solid state drives (SSDs) are the two main storage options commonly deployed in data centers. With HDD, data are stored magnetically in spinning disks, and data retrieval is slow; therefore, HDD is practical only for data that does not need to be accessed frequently. SSDs are a newer technology where data is stored in integrated circuits, which dramatically reduces access time. Because SSDs are more expensive than HDDs, a hybrid solution provides a good compromise between price and performance.

The NAS Lustre filesystems (/nobackupp[*17-19, 27-29*]) are configured using Progressive File Layout (PFL) with HDDs and SSDs configured as different Lustre pools. This article provides information about these filesystems, using /nobackupp17 as an example.

SSD and HDD Pools

To view a list of a filesystem's object storage targets (OSTs) showing whether they are in the HDD pool or the SSD pool, use the lfs pool_list command. The OSTs are shown in hexadecimal format:

```
pfe% lfs pool_list nbp17.hdd-pool
Pool: nbp17.hdd-pool
nbp17-OST0000_UUID
nbp17-OST0001_UUID
..
nbp17-OST001e_UUID
nbp17-OST001f_UUID

pfe% lfs pool_list nbp17.ssd-pool
Pool: nbp17.ssd-pool
nbp17-OST0064_UUID
nbp17-OST0065_UUID
...
nbp17-OST0082_UUID
nbp17-OST0083_UUID
```

The available SSD space in each filesystem is much smaller than the HDD space. In the example below, the lfs df command shows that the size of each /nobackupp17 HDD OST is 565.0 terabytes (TB), while each SSD OST is only 30.3 TB. The output also shows both the hexadecimal (far left) and decimal (far right) labels of each OST.

```
pfe% lfs df -h /nobackupp17 | grep OST
nbp17-OST0000_UUID      565.0T    322.7T    213.8T  61% /nobackupp17[OST:0]
nbp17-OST0001_UUID      565.0T    323.5T    213.0T  61% /nobackupp17[OST:1]
...
nbp17-OST001e_UUID      565.0T    322.9T    213.7T  61% /nobackupp17[OST:30]
nbp17-OST001f_UUID      565.0T    325.1T    211.4T  61% /nobackupp17[OST:31]
nbp17-OST0064_UUID       30.3T     23.5T      6.4T  79% /nobackupp17[OST:100]
nbp17-OST0065_UUID       30.3T     23.5T      6.4T  79% /nobackupp17[OST:101]
...
nbp17-OST0082_UUID       30.3T     23.5T      6.5T  79% /nobackupp17[OST:130]
nbp17-OST0083_UUID       30.3T     23.5T      6.5T  79% /nobackupp17[OST:131]
```

Configuration of the NAS Lustre Filesystems

The following table summarizes the configurations of /nobackupp[*17-19, 27-29*].

Filesystem	**nbp17**	**nbp18**	**nbp19**	**nbp27**	**nbp28**	**nbp29**
# of OSTs in HDD	32 (OST: 0-31)	32 (OST: 0-31)	16 (OST: 0-15)	32 (OST: 0-31)	32 (OST: 0-31)	16 (OST: 0-15)
Size/OST in HDD	565.0 T	565.0 T	565.0 T	565.0 T	565.0 T	565.0 T
Total Space in HDD	18.6 P	18.6 P	9.3 P	18.6 P	18.6 P	9.3 P
# of OSTs in SSD	32 (OST: 100-131)	16 (OST: 100-115)	16 (OST: 100-115)	32 (OST: 100-131)	16 (OST: 100-115)	16 (OST: 100-115)
Size/OST in SSD	30.3 T	60.5 T	29.6 T	30.3 T	60.5 T	29.6 T
Total Space in SSD	~ 1 P	~ 1 P	~ 0.5 P	~ 1 P	~ 1 P	~ 0.5 P

Progressive File Layout (PFL)

The Lustre filesystems take advantage of the Lustre PFL feature, which increases the stripe count of each file in a step-wise manner as the file grows in size. Specifically, PFL is built using composite layouts described by a series of components. Each component covers a non-overlapping portion of the file and has its own stripe setting in size, count, index and pool.

Default PFL Configurations

For each filesystem, a NAS-created default layout provides reasonable performance for a variety of file I/O patterns—and relieves users of the need to determine and set striping.

To find the default layout of each filesystem, use the lfs getstripe command as follows:

```
pfe% lfs getstripe -d /nobackupp17
lcm_layout_gen:    0
...
 lcme_extent.e_start: 0
 lcme_extent.e_end:   268435456
  stripe_count:  1    stripe_size: 16777216   pattern:   raid0   stripe_offset: -1  pool:  ssd-pool

 ...
 lcme_extent.e_start: 268435456
 lcme_extent.e_end:   5368709120
  stripe_count:  16   stripe_size: 16777216   pattern:   raid0   stripe_offset: -1  pool:  ssd-pool

 ...
 lcme_extent.e_start: 5368709120
 lcme_extent.e_end:   EOF
  stripe_count:  16   stripe_size: 16777216   pattern:   raid0   stripe_offset: -1  pool:  hdd-pool
```

The output shows that /nobackupp17 is configured with three components:

- The first component setting applies to the file segment from the beginning up to 268435456 B (256 MB), with a stripe count of 1 and stripe size of 16 MB; this component uses ssd-pool.
- The second component setting applies to the file segment from 256 MB to 5 GB with a stripe count of 16 and stripe size of 16 MB; this component uses ssd-pool.
- The last component setting applies to the file segment from 5 GB to the end of the file with a stripe count of 16 and stripe size of 16 MB; this component uses hdd-pool.

With this composite layout, I/O to and from the first 5 GB of files will benefit from the higher performance of SSDs.

Note: The default component layouts may be adjusted by NAS system administrators as needed.

Customized PFL Configurations

If you have experience using the PFL feature, and you want to have more control of your application's I/O, you can choose your own composite layouts using the lfs setstripe command with various options, such as --component-end (or -E), --stripe-size (or -S), --stripe-count (or -c), --overstripe-count (or -C), --pool (or -p). For example:

```
lfs setstripe \
   -E 256M -C 1  -S 1M -p ssd-pool \
   -E 4G   -C 4  -S 1M -p ssd-pool \
   -E EOF  -C 16 -S 1M -p hdd-pool \
directory_name_or_filename
```

Note: If not explicitly set, files inherit layouts from their parent directory.

WARNING: SSD space is limited. Therefore, do not set striping for very large files to use only the SSD pool.

Mirroring and Migration of Data from SSD to HDD

As the limited SSD space fills up, data that is stored in SSDs is migrated over to HDDs to free up SSD space for I/O by new jobs. This automatic migration is accomplished through a mirroring mechanism.

After a file is closed, SSD data is queued to be mirrored to the HDD. The data will be in dual state—stored on both the SSD and the HDD—until SSD usage reaches a certain point; then, the SSD copy will be removed to free up space. Typically, the oldest files are removed first when SSD usage reaches 80%, and other files are removed more aggressively when usage reaches 90%. The mirroring and migration settings may be adjusted by NAS system administrators as needed.

Note: Mirrored data that exists in both SSDs and HDDs are counted twice against your quota.

To find out if a file is in dual state or has been migrated completely to HDDs, use the lfs getstripe filename command, as shown in the next two examples.

Output Showing Data in Dual State

If the lfs getstripe filename output shows entries with different values for lcme_mirror_id, the data is in dual state, as illustrated in this example:

```
pfe% lfs getstripe file1
file1
  lcm_layout_gen:    5
  lcm_mirror_count:  2
  lcm_entry_count:   4
    lcme_id:             65537
    lcme_mirror_id:      1     <-------
    lcme_flags:          init,prefer
    lcme_extent.e_start: 0
    lcme_extent.e_end:   268435456
      lmm_stripe_count:  1
```

```
    lmm_stripe_size:   16777216
    lmm_pattern:       raid0
    lmm_layout_gen:    0
    lmm_stripe_offset: 128
    lmm_pool:          ssd-pool
    lmm_objects:
    - 0: { l_ost_idx: 128, l_fid: [0x800000409:0x413e0a:0x0] }

  lcme_id:             65538
  lcme_mirror_id:      1   <--------
  lcme_flags:          init,prefer
  lcme_extent.e_start: 268435456
  lcme_extent.e_end:   5368709120
    lmm_stripe_count:  16
    lmm_stripe_size:   16777216
    lmm_pattern:       raid0
    lmm_layout_gen:    0
    lmm_stripe_offset: 103
    lmm_pool:          ssd-pool
    lmm_objects:
    - 0: { l_ost_idx: 103, l_fid: [0x340000409:0x4305bb:0x0] }
    - 1: { l_ost_idx: 131, l_fid: [0x500000409:0x449f9f:0x0] }
    .....
    - 14: { l_ost_idx: 120, l_fid: [0x9c000040a:0x415384:0x0] }
    - 15: { l_ost_idx: 127, l_fid: [0x7c0000409:0x4100a6:0x0] }

  lcme_id:             65539
  lcme_mirror_id:      1  <---------
  lcme_flags:          init
  lcme_extent.e_start: 5368709120
  lcme_extent.e_end:   EOF
    lmm_stripe_count:  16
    lmm_stripe_size:   16777216
    lmm_pattern:       raid0
    lmm_layout_gen:    0
    lmm_stripe_offset: 27
    lmm_pool:          hdd-pool
    lmm_objects:
    - 0: { l_ost_idx: 27, l_fid: [0xfc0000400:0xd887b2:0x0] }
    - 1: { l_ost_idx: 26, l_fid: [0xb00000404:0xd8ed7c:0x0] }
    ....
    - 14: { l_ost_idx: 23, l_fid: [0xe40000400:0xd97afb:0x0] }
    - 15: { l_ost_idx: 9, l_fid: [0xe00000400:0x36d9a4:0x0] }

  lcme_id:             131073
  lcme_mirror_id:      2  <--------
  lcme_flags:          init
  lcme_extent.e_start: 0
  lcme_extent.e_end:   EOF
    lmm_stripe_count:  16
    lmm_stripe_size:   1048576
    lmm_pattern:       raid0
    lmm_layout_gen:    0
    lmm_stripe_offset: 9
    lmm_pool:          hdd-pool
    lmm_objects:
    - 0: { l_ost_idx: 9, l_fid: [0xe00000400:0x36d9bd:0x0] }
    - 1: { l_ost_idx: 19, l_fid: [0x120000040d:0xdaa1a0:0x0] }
    ....
    - 14: { l_ost_idx: 28, l_fid: [0xf40000400:0xd7c343:0x0] }
    - 15: { l_ost_idx: 11, l_fid: [0xc00000402:0x36d1eb:0x0] }
```

Output Showing Data Completely Migrated to HDDs

This example shows an older file that has been completely migrated to HDDs:

```
pfe% lfs getstripe file2
file2
  lcm_layout_gen:    2
  lcm_mirror_count:  1
  lcm_entry_count:   1
    lcme_id:             131073
    lcme_mirror_id:      2
    lcme_flags:          init
    lcme_extent.e_start: 0        <-----
    lcme_extent.e_end:   EOF      <-----
      lmm_stripe_count:  16
      lmm_stripe_size:   1048576
      lmm_pattern:       raid0
      lmm_layout_gen:    0
```

```
    lmm_stripe_offset: 24
    lmm_pool:          hdd-pool
    lmm_objects:
    - 0: { l_ost_idx: 24, l_fid: [0xd8000040e:0x475b02:0x0] }
    - 1: { l_ost_idx: 3, l_fid: [0x130000040e:0x4733cd:0x0] }
    ....
    - 14: { l_ost_idx: 20, l_fid: [0xc8000040e:0x482bcc:0x0] }
    - 15: { l_ost_idx: 18, l_fid: [0xe8000040e:0x47cc62:0x0] }
```

Note: When your file is completely migrated to HDD (removed from SDD), the (ctime) of the file will change to the creation time of the mirrored copy on HDD, as described below.

Change Time (ctime) Example

When a file is mirrored, a second copy of the data is created. The file will retain both sets of data until you break the mirror, or when the filesystem automatically flushes the SSD pool—at which time, the (ctime) will be updated to reflect the time the mirror was created.

In the following example, the stat command is used to show the time metadata of a test file that was created at 15:48:45:

```
% stat testfile
  File: 'testfile'
  Size: 59535      Blocks: 120        IO Block: 4194304 regular file
Device: 521e79ech/1377729004d Inode: 342273918398214817  Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2022-02-14 15:48:45.000000000 -0800
Modify: 2022-02-14 15:48:45.000000000 -0800
Change: 2022-02-14 15:48:45.000000000 -0800
```

At time 15:50:27, the file is mirrored but the ctime remains unchanged:

```
% stat testfile
  File: 'testfile'
  Size: 59535      Blocks: 120        IO Block: 4194304 regular file
Device: 521e79ech/1377729004d Inode: 342273918398214817  Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2022-02-14 15:48:45.000000000 -0800
Modify: 2022-02-14 15:48:45.000000000 -0800
Change: 2022-02-14 15:48:45.000000000 -0800
 Birth: -

pfe25 /nobackupp18/username % lfs getstripe testfile
testfile
  lcm_layout_gen:    1
  lcm_mirror_count:  2
  lcm_entry_count:   4
    lcme_id:             65537
    lcme_mirror_id:      1
    lcme_flags:          init,prefer
    lcme_extent.e_start: 0
    lcme_extent.e_end:   268435456
      lmm_stripe_count:  1
      lmm_stripe_size:   16777216
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: 109
      lmm_pool:          ssd-pool
      lmm_objects:
      - 0: { l_ost_idx: 109, l_fid: [0x90000040a:0x53990:0x0] }

    lcme_id:             65538
    lcme_mirror_id:      1
    lcme_flags:          prefer
    lcme_extent.e_start: 268435456
    lcme_extent.e_end:   5368709120
      lmm_stripe_count:  -1
      lmm_stripe_size:   16777216
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: -1
      lmm_pool:          ssd-pool

    lcme_id:             65539
    lcme_mirror_id:      1
    lcme_flags:          0
    lcme_extent.e_start: 5368709120
    lcme_extent.e_end:   EOF
      lmm_stripe_count:  16
      lmm_stripe_size:   16777216
      lmm_pattern:       raid0
      lmm_layout_gen:    0
```

```
    lmm_stripe_offset: -1
    lmm_pool:          hdd-pool

  lcme_id:             131073
  lcme_mirror_id:      2
  lcme_flags:          init
  lcme_extent.e_start: 0
  lcme_extent.e_end:   EOF
    lmm_stripe_count:  1
    lmm_stripe_size:   1048576
    lmm_pattern:       raid0
    lmm_layout_gen:    0
    lmm_stripe_offset: 0
    lmm_pool:          hdd-pool
    lmm_objects:
    - 0: { l_ost_idx: 0, l_fid: [0x2c0000409:0x30c3e:0x0] }
```

Later, the file is flushed from the SSD. At that time, the file's ctime is updated to reflect the creation time of the copy on HDD:

```
% stat testfile
  File: 'testfile'
  Size: 59535      Blocks: 120        IO Block: 4194304 regular file
Device: 521e79ech/1377729004d Inode: 342273918398214817  Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2022-02-14 15:50:27.000000000 -0800
Modify: 2022-02-14 15:48:45.000000000 -0800
Change: 2022-02-14 15:50:27.000000000 -0800
 Birth: -
```

References

- Progressive File Layout
- File Level Redundancy
- Layout Enhancement High Level Design

At NAS, Lustre (/nobackup) filesystems are shared among many users and many application processes, which can cause contention for various Lustre resources. This article explains how Lustre I/O works, and provides best practices for improving application performance.

How Does Lustre I/O Work?

When a client (a compute node from your job) needs to create or access a file, the client queries the metadata server (MDS) and the metadata target (MDT) for the layout and location of the file's stripes. Once the file is opened and the client obtains the striping information, the MDS is no longer involved in the file I/O process. The client interacts directly with the object storage servers (OSSes) and object storage targets (OSTs) to perform I/O operations such as locking, disk allocation, storage, and retrieval.

If multiple clients try to read and write the same part of a file at the same time, the Lustre distributed lock manager enforces coherency so that all clients see consistent results.

Jobs being run on NAS compute systems contend for shared resources in NAS's Lustre filesystem. Each server that is part of a Lustre filesystem can only handle a limited number of I/O requests (read, write, stat, open, close, etc.) per second. An excessive number of such requests, from one or more users and one or more jobs, can lead to contention for storage resources. Contention slows the performance of your applications and weakens the overall health of the Lustre filesystem. To reduce contention and improve performance, please apply the examples below to your compute jobs while working in our high-end computing environment.

Best Practices

Avoid Using ls -l

The ls -l command displays information such as ownership, permission, and size of all files and directories. The information on ownership and permission metadata is stored on the MDTs. However, the file size metadata is only available from the OSTs. So, the ls -l command issues Remote Procedure Calls (RPCs) to the MDS/MDT and OSSes/OSTs for every file/directory to be listed. RPC requests to the OSSes/OSTs are very costly and can take a long time to complete if there are many files and directories.

- Use ls by itself if you just want to see if a file exists
- Use ls -l *filename* if you want the long listing of a specific file

Avoid Having a Large Number of Files in a Single Directory

Opening a file keeps a lock on the parent directory. When many files in the same directory are to be opened, it creates contention. A better practice is to split a large number of files (in the thousands or more) into multiple subdirectories to minimize contention.

Avoid Accessing Small Files on Lustre Filesystems

Accessing small files on the Lustre filesystem is not efficient. When possible, keep them on an NFS-mounted filesystem (such as your NAS compute resources home filesystem /u/*username*) or copy them from Lustre to /tmp on each node at the beginning of the job, and then access them from /tmp.

Keep Copies of Your Source Code on the NAS Compute Resources Home Filesystem and/or Lou

Be aware that files under /nobackup are *not* backed up. Make sure that you save copies of your source codes, makefiles, and any other important files on your NAS compute resources home filesystem. If your NAS compute resources home directory quota isn't large enough to keep all of these files, you can request a larger quota and/or create tarred copies of these files on Lou.

Avoid Accessing Executables on Lustre Filesystems

There have been a few incidents on NAS compute systems where users' jobs encountered problems while accessing their executables on the /nobackup filesystem. The main issue is that the Lustre clients can become unmounted temporarily when there is a very high load on the Lustre filesystem. This can cause a bus error when a job tries to bring the next set of instructions from the inaccessible executable into memory.

Executables run slower when run from the Lustre filesystem. It is best to run executables from your NAS compute resources home filesystem. On rare occasions, running executables from the Lustre filesystem can cause executables to be corrupted. Avoid copying new executables over existing ones of the same name within the Lustre filesystem. The copy causes a window of time (about 20 minutes) where the executable will not function. Instead, the executable should be accessed from your home filesystem during runtime.

Limit the Number of Processes Performing Parallel I/O

Given that the numbers of OSSes and OSTs of various NAS Lustre filesystems are about a hundred or fewer, there will be contention if a large number of processes of an application are involved in parallel I/O. Instead of allowing all processes to do the I/O, choose

just a few processes to do the work. For writes, these few processes should collect the data from other processes before the writes. For reads, these few processes should read the data and then broadcast the data to others.

Understand the Effect of Stripe Counts/Sizes for MPI Collective Writes

For programs that call MPI collective write functions, such as MPI_File_write_all, MPI_File_write_at_all, and MPI_File_write_ordered, it is important to understand the effect of stripe counts on performance.

Background

MPI I/O supports the concept of collective buffering. For some filesystems, when multiple MPI processes are writing to the same file in a coordinated manner, it is much more efficient for the different processes to send their writes to a subset of processes in order to do a smaller number of bigger writes. By default, with collective buffering, the write size is set to be the same as the stripe size of the file.

With Lustre filesystems, there are two main factors in the HPE (SGI) MPT algorithm that chooses the number of MPI processes to do the writes: the stripe count and number of nodes. When the number of nodes is greater than the stripe count, the number of collective buffering processes is the same as the stripe count. Otherwise, the number of collective buffering processes is the largest integer less than the number of nodes that evenly divides the stripe count. In addition, MPT chooses the first rank from the first *n* nodes to come up with *n* collective buffering processes.

Note: Intel MPI behaves similarly to HPE (SGI) MPT on Lustre filesystems.

Enabling Collective Buffering Automatically

You can let each MPI implementation enable collective buffering for you, without any code changes.

HPE (SGI) MPT automatically enables collective buffering for the collective write calls using the algorithm described above. This method requires no changes in the user code or in the mpiexec command line. For example, if the stripe count is 1, only rank 0 does the collective writes, which can result in poor performance. Therefore, experimenting with different stripe counts on the whole directory and/or individual files is strongly recommended.

Intel MPI also does collective buffering, similar to HPE (SGI) MPT, when the I_MPI_EXTRA_FILESYSTEM and I_MPI_EXTRA_FILESYSTEM_LIST variables are set appropriately, as follows:

```
mpiexec.hydra  -env I_MPI_EXTRA_FILESYSTEM on \
      -env  I_MPI_EXTRA_FILESYSTEM_LIST lustre \
      -np xx a.out
```

Enabling Collective Buffering via Code Changes

In this method, you provide "hints" in the source code to inform MPI what to do with specific files. For example:

```
  call MPI_Info_create(info, status)
  call MPI_Info_set(info, "romio_cb_write", "enable", STATUS)
  call MPI_Info_set(info, "striping_unit", "1048576", STATUS)
  call MPI_Info_set(info, "striping_factor", "16", STATUS)
  ...
  call MPI_File_open(MPI_COMM_WORLD, file_name, MPI_MODE_WRONLY, info, unit, status)
```

Note: The hints are only advisory and may not be honored. For example, SGI MPT 2.12r26 honors these hints, but MPT 2.14r19 does not. Intel MPI 5.0x honors these hints when the I_MPI_EXTRA_FILESYSTEM and I_MPI_EXTRA_FILESYSTEM_LIST variables are set appropriately, as follows:

```
mpiexec.hydra  -env I_MPI_EXTRA_FILESYSTEM on \
      -env  I_MPI_EXTRA_FILESYSTEM_LIST lustre \
      -np xx a.out
```

Stripe Align I/O Requests to Minimize Contention

Stripe aligning means that the processes access files at offsets that correspond to stripe boundaries. This helps to minimize the number of OSTs a process must communicate for each I/O request. It also helps to decrease the probability that multiple processes accessing the same file communicate with the same OST at the same time.

One way to stripe-align a file is to make the stripe size the same as the amount of data in the write operations of the program.

Avoid Repetitive "stat" Operations

Some users have implemented logic in their scripts to test for the existence of certain files. Such tests generate "stat" requests to the Lustre server. When the testing becomes excessive, it creates a significant load on the filesystem. A workaround is to slow down the testing process by adding sleep in the logic. For example, the following user script tests the existence of the files WAIT and STOP to decide what to do next.

```
touch WAIT
 rm STOP
```

```
while ( 0 <= 1  )
 if(-e WAIT) then
   mpiexec ...
   rm WAIT
 endif
 if(-e STOP) then
   exit
 endif
end
```

When neither the WAIT nor STOP file exists, the loop ends up testing for their existence as quickly as possible (on the order of 5,000 times per second). Adding sleep inside the loop slows down the testing.

```
touch WAIT
 rm STOP

 while ( 0 <= 1  )
  if(-e WAIT) then
    mpiexec ...
    rm WAIT
  endif
  if(-e STOP) then
    exit
  endif
  sleep 15
 end
```

Avoid Having Multiple Processes Open the Same File(s) at the Same Time

On Lustre filesystems, if multiple processes try to open the same file(s), some processes will not able to find the file(s) and your job will fail.

The source code can be modified to call the sleep function between I/O operations. This will reduce the occurrence of multiple, simultaneous access attempts to the same file from different processes.

```
100  open(unit,file='filename',IOSTAT=ierr)
     if (ierr.ne.0) then
      ...
     call sleep(1)
     go to 100
     endif
```

When opening a read-only file in Fortran, use ACTION='read' instead of the default ACTION='readwrite'. The former will reduce contention by not locking the file.

```
open(unit,file='filename',ACTION='READ',IOSTAT=ierr)
```

Avoid Repetitive Open/Close Operations

Opening files and closing files incur overhead and repetitive open/close should be avoided.

If you intend to open the files for read only, make sure to use ACTION='READ' in the open statement. If possible, read the files once each and save the results, instead of reading the files repeatedly.

If you intend to write to a file many times during a run, open the file once at the beginning of the run. When all writes are done, close the file at the end of the run.

See Lustre Basics for more information.

Use the Soft Link to Refer to Your Lustre Directory

Your /nobackup directory is created on a specific Lustre filesystem, such as /nobackupp17 or /nobackupp18, but you can use a soft link to refer to the directory no matter which filesystem it is on:

/nobackup/*your_username*

By using the soft link, you can easily access your directory without needing to know the name of the underlying filesystem. Also, you will not need to change your scripts or re-create any symbolic links if a system administrator needs to migrate your data from one Lustre filesystem to another.

Preserve Corrupted Files for Investigation

When you notice a corrupted file in your /nobackup directory, it is important to preserve the file to allow NAS staff to investigate the cause of corruption. To prevent the file from being accidentally overwritten or deleted by your scripts, we recommend that you rename the corrupted file using:

pfe% mv *filename filename*.corrupted

Note: Do not use cp to create a new copy of the corrupted file.

Report the problem to NAS staff by sending an email to support@nas.nasa.gov. Include how, when, where the corrupted file was generated, and anything else that may help with the investigation.

See the following articles for more information:

- Lustre Basics - Setting Stripe Parameters
- Shift Command Options

Reporting Problems

If you report performance problems with a Lustre filesystem, please be sure to include the time, hostname, PBS job number, name of the filesystem, and the path of the directory or file that you are trying to access.Your report will help us correlate issues with recorded performance data to determine the cause of efficiency problems.