

# Lustre Best Practices

## Category: Lustre on Pleiades

**Summary:** At NAS, Lustre filesystems (/nobackup) are shared among many users and many application processes, which causes 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 Pleiades contend for shared resources in NAS's Lustre filesystem. The Lustre server can only handle about 15,000 remote procedure calls (RPCs, inter-process communications that allow the client to cause a procedure to be executed on the server) per second. 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 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 home filesystem on Pleiades `/u/username`) or copy them from Lustre to `/tmp` on each node at the beginning of the job, and then access them from `/tmp`.

## Use a Stripe Count of 1 for Directories with Many Small Files

If you must keep small files on Lustre, be aware that `stat` operations are more efficient if each small file resides in one OST. Create a directory to keep small files, set the stripe count to 1 so that only one OST will be needed for each file. This is useful when you extract source and header files (which are usually very small files) from a tarfile. Use the Lustre utility `lfs` to create a specific striping pattern, or find the striping pattern of existing files.

```
pfel% mkdir dir_name
pfel% lfs setstripe -s 1m -c 1 dir_name
pfel% cd dir_name
pfel% tar -xf tarfile
```

If there are large files in the same directory tree, it may be better to allow them to stripe across more than one OST. You can create a new directory with a larger stripe count and copy the larger files to that directory. Note that moving files into that directory with the `mv` command will not change the strip count of the files. Files must be *created in* or *copied* to a directory to inherit the stripe count properties of a directory.

```
pfel% mkdir dir_count_4
pfel% lfs setstripe -s 1m -c 4 dir_count_4
pfel% cp file_count_1 dir_count_4
```

If you have a directory with many small files (less than 100 MB) and a few very large files (greater than 1 GB), then it may be better to create a new subdirectory with a larger stripe count. Store just the large files and create symbolic links to the large files using the `symlink` command `ln`.

```
pfel% mkdir bigstripe
pfel% lfs setstripe -c 16 -s 4m bigstripe
pfel% ln -s bigstripe/large_file large_file
```

## Use mtar for Creating or Extracting a tar file

A modified `gnu tar` command, `/usr/local/bin/mtar`, is Lustre stripe aware and will create tar files or extract files with appropriately sized stripe counts. Currently, the number of stripes is set to the number of gigabytes of the file.

## Keep Copies of Your Source Code on the Pleiades Home Filesystem and/or Lou

Be aware that files under `/nobackup[p1-p6]` are *not* backed up. Make sure that you have copies of your source codes, makefiles, and any other important files saved on your Pleiades home filesystem or on Lou, the NAS storage system.

## Avoid Accessing Executables on Lustre Filesystems

There have been a few incidents on Pleiades 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 home filesystem on Pleiades. 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.

## Increase the `stripe_count` for Parallel Writes to the Same File

When multiple processes are writing blocks of data to the same file in parallel, the I/O performance for large files will improve when the `stripe_count` is set to a larger value. The stripe count sets the number of OSTs the file will be written to. By default, the stripe count is set to 1. While this default setting provides for efficient access of metadata--for example to support the `ls -l` command--large files should use stripe counts of greater than 1. This will increase the aggregate I/O bandwidth by using multiple OSTs in parallel instead of just one. A rule of thumb is to use a stripe count approximately equal to the number of gigabytes in the file.

Another good practice is to make the stripe count be an integral factor of the number of processes performing the write in parallel, so that you achieve load balance among the OSTs. For example, set the stripe count to 16 instead of 15 when you have 64 processes performing the writes.

## Limit the Number of Processes Performing Parallel I/O

Given that the numbers of OSSes and OSTs on Pleiades 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.

## 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 be 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 also: [Lustre Basics](#)

## 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.

---

Article ID: 226

Last updated: 26 Sep, 2012

Computing at NAS -> Best Practices -> Lustre on Pleiades -> Lustre Best Practices

<http://www.nas.nasa.gov/hecc/support/kb/entry/226/?ajax=1>