Lustre best practices

Distributed filesystems such as Lustre are ideal for HPC and HTC environments. In these environments the typical workload consists of large files that have to be accessed from many compute nodes with very high bandwidth and/or low latency. Therefore these filesystems are very different from the filesystems used on desktop computers or isolated servers. Although they excel at handling large files, they also have strong limitations when handling small files and access patterns more commonly found in enterprise and desktop environments. Operations that can be extremely fast on a workstation local disk can be painfully slow and expensive on a Lustre filesystem, affecting both the users performing those operations and eventually all other users. These best practices and recommendations are aimed to enable a smooth use of Lustre by minimizing or avoiding unnecessary or very expensive filesystem operations.

Avoid accessing attributes of files and directories

Accessing metadata information such as file attributes (e.g. type, ownership, protection, size, dates, etc) in Lustre is resource intensive and can degrade the filesystem performance, especially when performed frequently or over large directories. Minimize the use of system calls that access or modify these attributes such as `stat()`, `statx()`, `open()`, `openat()`, etc.

The same applies to commands like `ls -l` or `ls --color` that make use of above mentioned calls. Instead use a simple `ls` or `ls -l filename`.

Avoid using commands that access metadata massively

Avoid using commands such as `ls -R`, `find`, `locate`, `du`, `df` and similar. These commands walk the filesystem recursively and/or perform heavy metadata operations. They are very intensive on accessing filesystem metadata and can degrade badly the overall file system performance. If walking the filesystem recursively is absolutely required, then use the Lustre provided `lfs find` instead of `find` and similar tools.

Use the Lustre lfs command

To minimize the number of Lustre RPC calls, whenever possible use the `lfs` commands instead of the system provided commands:

- `lfs df` => instead of `df`
- `lfs find` => instead of `find`

Avoid using wild cards

Expanding the wild cards is resource intensive. Executing commands with wildcards on a very large numbers of files may take a very long time and badly impact the filesystem performance. Instead

of using wild cards, create a list of the target files and apply the command to each of these files.

Read Only Access

Whenever possible open the files as read-only using `O_RDONLY`, furthermore if you don't need to update the file access time then open the files as `O_RDONLY | O_NOATIME`. If access time information is needed while performing parallel I/O then, let the master process open the files as `O_RDONLY` and all other ranks open the same files as `O_RDONLY|O_NOATIME`.

Avoid having a large number of files in a single directory

When a file is accessed, Lustre places a lock on the parent directory. When many files in the same directory are to be opened this creates contention. Writing thousands of files to a single directory produces massive load on Lustre metadata servers, often resulting on taking filesystems offline. Accessing a single directory containing thousands of files can cause heavy resource contention degrading the filesystem performance.

The alternative is to organize the data into multiple sub-directories and split the files across them. A common approach is to use the square root of the number of files, for instance for 90000 files the square root would be 300, therefore 300 directories should be created containing 300 files each.

Avoid small files

Accessing small files on the Lustre filesystem is very inefficient. The recommended file size is above 1GB. Reorganize the data in large files or use file formats such as **HDF5** or **NetCDF**. Alternatively if the total size of the files is small such as a few gigabytes then copy the small files to `/tmp` or to a scratch directory local to each compute node at the beginning of the job (do not forget to transfer and/or delete the files in the end). This approach can be combined with the use of archival tools such as `tar` to store the small files in one or more large tarballs that can be kept on Lustre more efficiently. When data is read-only, another alternative is to create a disk image and mount it read-only through loopback in each cluster node as described in (ref 4). Container tools such as `singularity` can also enable the use of loopback mounted disk images.

Avoid small buffer sizes

When reading or writing to files, Lustre performs much better with large buffer sizes (>= 1MB). Aggregating small read and write operations into larger ones is highly recommended. MPI-IO Collective Buffering enables aggregate I/O.

Avoid small repetitive file operations

Avoid performing repetitive small I/O operations such as frequently opening files in append mode, write small amounts of data, and close the file. Instead open the file once perform all I/O operations and close.

Avoid multiple processes opening the same files at the same time

Multiple processes opening the same files at the same time can create contention and file open errors. Instead perform the open from a single process (master), or open the file read-only to avoid locking, or implement the open with a try and error approach with a sleep in case of error.

Avoid accessing the same file region from many processes

If multiple processes access the same file region at the same time, the Lustre distributed lock manager enforces coherency so that all clients see consistent results. Having many processes trying to access the same file region simultaneously can cause performance degradation.

In this case it might be preferable to either: replicate the file, split the file, perform the I/O operations from a single process rank or make sure simultaneous access will not occur. In any case it is recommended to keep the amount of file-open and file-lock operations in parallel as small as possible to reduce contention.

Appending to the same file from many processes

This is similar to the previous recommendation. If multiple processes try to append to the same file this will trigger locking and this can cause heavy contention. Ideally only one process should append each file.

File operations through the master

When accessing small shared files in a parallel job it is often more efficient to perform all the required operations through the master process and if needed broadcast the data to the other ranks, instead of accessing the same files from every rank. Similarly, if multiple ranks of a parallel job require information regarding a given file, the most efficient approach is to have the master process performing the required calls (e.g. `stat()`, `fstat()`, etc) and then broadcast the information to the other ranks. For an example see “Broadcast Stat” in (ref 3).

File Striping

In Lustre large files can be split into segments that in turn can be automatically spread across multiple storage devices. File striping is useful for parallel I/O over large files. For this to work the mount point in question must be backed by multiple storage devices (OSTs). The command lfs df can be used to verify if a given mount point is backed by multiple OSTs. To obtain file striping information for a given file use:

- `lfs getstripe filename`

The file striping can be set using the command `lfs setstripe`. If the command is applied to a directory it will define the default stripe settings for files created in that directory. A subdirectory inherits all stripe settings from its parent directory. If the command is applied to a file it will stripe that file across OSTs according to the specified settings.

- `lfs setstripe -s  64m -c 4 filename` => split filename in 64MB segments and spread it across 4 OSTs

If a large file is to be shared in parallel by several ranks (processes) having each rank working on its own portion of the file, then it might be useful to stripe the file in a number of segments equal to the number of processes, or a multiple of the number of processes.

For maximum performance I/O requests should be stripe aligned, this means that the processes accessing the file should do it at offsets that match the stripe boundaries. This minimizes the chances of a process having to access more than one segment (and more than one OST) to obtain the required data.

For small files striping should be disabled, this can be achieved by setting a stripe count of 1. The same applies if a large file will be accessed by a single process.

- `lfs setstripe -s 1m -c 1 mydirectory/smallfilesdir/`

Avoid installing software on Lustre

Software is usually composed of many small files and as previously mentioned accessing many small files on Lustre can place a huge load on metadata servers. The software compilations in particular can be better performed locally by copying or untaring the software to the `/tmp/$USER/` of a login server.

Furthermore, under high load I/O access to the Lustre file systems may block. If executables are stored in Lustre and access to the filesystem fails the executables may crash. Therefore, whenever possible, it is better to copy the executables to the `/tmp` of the cluster nodes.

To address these issues the INCD compute environment includes a special read-only caching filesystem (CVMFS from CERN) that provides scalable software distribution within and across compute clusters. Contact the INCD support if deployment of software on the INCD CVMFS is desired.

File locking with flock

If applications are carefully designed the use of flock is not strictly necessary since Lustre can keep non-overlapping writes consistent, and can handle concurrent file append operations. Depending on the Lustre version the use of flocks may have a performance impact.

Backups

Perform regular backups of your data to a safe location. INCD does not perform data backups.

References

These best practices have been compiled from the INCD experience and from the following sources:

1. https://www.nas.nasa.gov/hecc/support/kb/lustre-best-practices_226.html
2. https://hpcf.umbc.edu/general-productivity/lustre-best-practices/

3. http://researchcomputing.github.io/meetup_fall_2014/pdfs/fall2014_meetup10_lustre.pdf
4. http://www.prace-ri.eu/IMG/pdf/WP245.pdf
5. https://opus.nci.org.au/display/Help/Lustre Best Practices#LustreBestPractices-7)LimittheNumberofProcessesPerformingParallelI/O
6. https://wiki.gsi.de/foswiki/bin/view/Linux/LustreFs

Revision #8
Created 2019-02-20 17:43:32 UTC by João Pina
Updated 2020-02-20 16:15:13 UTC by Jorge Gomes