WHITE PAPER

Intel® Enterprise Edition for Lustre* Software
High-Performance Data Division

Data Migration with Intel® Enterprise Edition for Lustre* Software

Best practices and recommendations from Intel's own testing and experience

As the tide of data rises ever higher, enterprises need to migrate their data from outdated, inefficient file systems to a more powerful file system like Intel® Enterprise Edition for Lustre* software.

Executive Summary

Data migration is a key consideration during the introduction of a new high-performance computing (HPC) storage platform. It can be a tedious experience to migrate terabytes or even petabytes of data from one storage platform to another. Data migration involves multiple challenges including (but not limited to) preserving permissions, ownerships, and metadata attributes and preventing data corruption.

This document can help system integrators and administrators choose an approach to HPC data migration. It also provides some planning tips, describes three useful reference architectures for moving HPC data to Intel® Enterprise Edition for Lustre* (Intel® EE for Lustre*) software, and evaluates two open source data migration tools.[1]

As the tide of data rises ever higher, enterprises need to migrate their data from outdated, inefficient file systems to a more powerful file system like Intel EE for Lustre. This document helps make that process easier and more efficient.

Table of Contents

Author

Chakravarthy Nagarajan
Solutions Architect, Intel

Background

High-performance computing (HPC) clusters generate large amounts of data that need to be effectively managed and preserved. The best practices and recommendations in this document can help system architects migrate HPC data from any POSIX (Portable Operating System Interface)-compliant file system to the Lustre* file system.

Although block-based data migration may seem attractive, it presents many challenges, including establishing interoperability between the existing and the new storage environments. This document focuses on the alternative option: file-based migration. With this approach, files are migrated using a variety of tools and techniques. File-based migration is more prevalent in the industry because it makes it easier to track migration progress. On the other hand, handling the files individually can be difficult.

Data migration needs effective planning and involves additional hardware and third-party software tools that are not included as part of Intel® Enterprise Edition for Lustre* (Intel® EE for Lustre*) software. It is important to understand the state of the data within the existing storage, such as how much of it is changing and which user directories are being used actively. It is necessary to examine the data files for permissions, ownership, time stamps, and xattr (extended attributes). The objective is to migrate data safely, along with the attributes that need to be preserved.

The following sections discuss choosing a data migration approach, planning considerations, data migration reference architectures, an evaluation of two data migration tools, and a discussion of other tools worthy of investigation. Table 1 explains some data migration terminology used in this document.

Note: The best practices and recommendations, tools, and techniques covered in this document are specific to the Lustre file system (that is, archival or hierarchical storage management solutions are out of scope).

Choosing a Data Migration Approach

As more data becomes available, driven by new data sources like social media and the Internet of Things (IoT), the demand for storage is growing exponentially in the HPC industry. Enterprises wishing to stay ahead of storage requirements often find themselves needing to upgrade an outdated storage platform to a more powerful, parallel storage platform such as Intel EE for Lustre software.

This document discusses two data migration approaches:

- **Offline or one-time data migration.** This approach is used when the data on the outdated storage platform is migrated to a new high-performance storage platform during the deployment of a new HPC infrastructure. In this scenario, the source could be any POSIX-compliant file system including Lustre while the destination is the Lustre file system. Although the migration happens only once, it is crucial to complete it in a timely manner without losing any data.
- **Live migration or disaster recovery.** Live migration of data is driven by policies that involve the critical availability of data during failure, as a subset of business continuity. In this scenario, both source and destination are the Lustre file system.

Planning for Data Migration

As mentioned earlier in this document, the planning stage is very crucial to make the data migration efficient and successful. Much of the data migration planning process is similar for both one-time and live migration. However, each approach does require specific considerations, as discussed in the following sections.

Offline or One-Time migration

Following are the factors to be considered while planning a one-time data migration:

- **Data movers.** It is important to understand the data migration performance requirements and service-level agreement (SLA) governing data availability on the destination (new) storage platform.
  - Based on the source and destination storage platform performance, determine the number of data movers that will yield the best performance from both source and destination.
  - It is recommended to run a benchmark, such as IOR or IOZONE, on the data movers to characterize the file system performance on both source and destination.
- **Data migration tool.** Choose a robust data migration tool, since it is one of the critical components of the data migration process.

Table 1. Data Migration Terminology

System Component	Definition
Data mover(s)	Server(s) or Lustre* client(s) responsible for moving the data between source and destination storage, running the required Linux* distribution as determined by the data migration tool, Lustre clients, interconnects, and so on.
Data migration tool	Software installed on the data movers which partitions and moves the data between the source and destination.
Source	**One-time migration:** An existing Lustre platform or an outdated Portable Operating System Interface (POSIX)-compliant file system. **Live migration or disaster recovery:** High-performance storage running Intel® Enterprise Edition for Lustre* software.
Destination	High-performance storage running Intel Enterprise Edition for Lustre software.
High-performance data network/fabric	The network/fabric on which the source and destination file systems are mounted on the data movers, including InfiniBand*, Intel® Omni-Path Architecture, 10GbE, or 1GbE.

- **Bandwidth and latency.** The bandwidth and latency of the network/fabric through which the source and the destination file systems are mounted play a key role in data migration performance.
  - Tune the kernel and the Lustre Networking (LNET) layer for the best performance.
  - The LNET self-test is a good utility to measure the LNET performance for tuning.
  - Refer to the LNET tuning guide for efficient tuning between multiple networks and fabrics.
- **Data bucketing.** The following tips can help improve migration performance:
  - Segregate the data based on size, and apply necessary Lustre tuning based on the file size for optimum performance.
  - Use find, lfs find or fpart for size-based segregation of data and prepare the list of files with various partitions.
  - Assign specific data movers to handle smaller files and others to handle larger files.
- **Object storage target (OST) pools and striping.** Use the following recommendations to further enhance migration performance:
  - If required, configure OST pools for dedicated bandwidth and better control of performance with respect to file size.
  - Apply striping for larger files.
- **Additional considerations.** The following recommendations can help avoid conflicts and failures:
  - Ensure both file systems are offline from production to yield the best performance. It is recommended to mount the source file system as read-only to avoid conflicts.
  - Running a batch job through a workload manager in re-queue on fail mode is strongly recommended for automatic failure handling.
  - Data validation can be done by checksum verification between source and destination files.
  - If the source file system is GPFS* (General Parallel File System), then consider mounting the source file system using CNFS (Clustered NFS) to avoid any conflicts between GPFS and the Lustre clients at the data movers.

Live Migration or Disaster Recovery

Apart from the typical data center-to-disaster recovery site requirements for hardware and software, the following factors can improve data migration performance between the data center and the disaster recovery site.

- **Data movers.** Choose the number of data movers based on the data migration performance requirements and the SLA governing data availability from the disaster recovery site.
- **Data migration tool.** Choose a robust data migration tool, since it is one of the critical components of the data migration process.
- **Lustre changelogs.** Consider implementing changelog-based disaster recovery replication, which records events that change the file system namespace or file metadata.
  - Changes such as file creation, deletion, renaming, and attribute changes are recorded with the target and parent file identifiers (FIDs), the name of the target, and a timestamp.
  - This approach helps reduce the metadata overhead and replicates only the changed data instead of traversing the entire file system tree.
- **Compression.** If required, enable compression for better disk space utilization of the disaster recovery site. Intel EE for Lustre software with ZFS[2] as the backend file system, enabled with the compression feature, may be a better choice for efficient space management.
- **Bandwidth and latency.** The network/fabric bandwidth and latency between the data center and the disaster recovery site plays a key role on the performance.
  - Tune the kernel and the LNET layer for the best performance.
  - The LNET self-test is a good utility to measure the performance for tuning. Refer to the LNET tuning guide for efficient tuning between multiple networks and fabrics.

Data Migration Architecture

The data migration architecture differs depending on how the source, destination, and data movers are configured. The following sections provide architecture information for the following three configurations:

- **Typical architecture.** The source, destination, and data movers are connected to the same fabric/network.
- **Multi-homed data movers.** The source is on a different network/fabric than the destination and the data movers use two separate connections.
- **LNET routers.** The source is on a different network/fabric than the destination and the data movers mount the source file system through the LNET routers.

Reference Architecture – Typical

Figure 1 illustrates the typical configuration where the source, destination, and the data movers are connected to the same network/fabric.

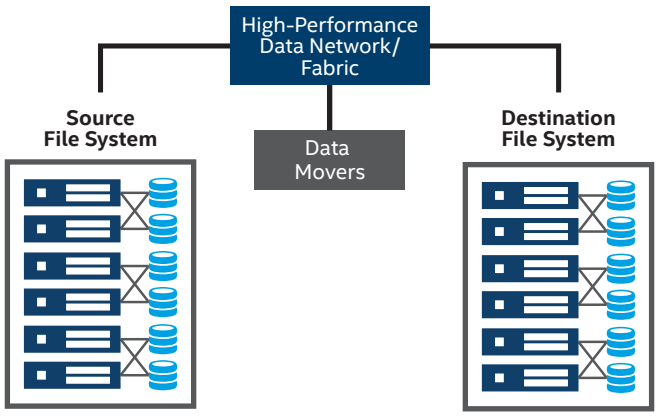

Figure 1. This shows the typical configuration where the source, destination, and the data movers are connected to the same network/fabric.

Reference Architecture – Multi-Homed Data Movers

Figure 2 illustrates the connectivity between the data movers, source, and destination if the source is on a different network/fabric than the destination. In this architecture, the data movers are connected to both networks/fabrics and mount the source file system over "High-Performance Data Fabric/Network 1" and the destination file system over "High-Performance Data Fabric/Network 2." This architecture is preferred because it is less complex and most efficient. However, compatibility issues may exist between the Lustre client, operating system version, Host Channel Adapters, Host Fabric Interface, and relevant drivers on the data movers.

Reference Architecture - LNET Routers

Figure 3 shows another approach to connectivity between the data movers, source, and destination if the source is on a different network/fabric than the destination. In this architecture, the data movers mount the source file system through the LNET routers, which are connected to both networks/fabrics. Refer to "Intel® Omni-path Storage Router – Design Guide" for more information on the detailed configuration.

Evaluation of Data Migration Tools

Multiple open source data migration tools are available, including cp, rsync, tar, GridFTP, bbcp, and bbftp. We have evaluated only fpart and Multi-Threaded Copy Program (MCP), which are parallel data movers that provide high efficiency, metadata preservation, data integrity, and robustness.

We used two types of data sets in our evaluations:

- **Small data set.** 100,000 * 16-MB files
- **Large data set.** One hundred * 50-GB files

The evaluations involved running several tests, the results of which are not representative of a production Lustre file system environment. Instead, they merely provide baseline comparisons between different methods and software applications.

Table 2. Test Environment

System Component	Configuration
Data movers	• 2x Intel® Xeon® processor E5-2660 v2 with 64 GB memory and 1x FDR InfiniBand* port • OS: CentOS*[3] Linux* release 7.2.1511 • Lustre Client: Intel® Enterprise Edition for Lustre* software 3.0
Data migration tools	• fpart and Multi-Threaded Copy Program
Source	• Storage: Dell PowerVault* MD3420 for metadata target (MDT) and management target (MGT) and 2x Dell PowerVault MD3460 for object storage targets (OSTs) • Useable Capacity: 261 TB • File system: Intel Enterprise Edition for Lustre software 2.4 with ldiskfs
Destination	• Storage: HPE Apollo* 4520 Appliance • Useable Capacity: 145 TB • File system: Intel Enterprise Edition for Lustre software 3.0 with ZFS*
High-performance data network/fabric	• 56-Gbps FDR InfiniBand*

Fpart Evaluation

Rsync is an open source data synchronization tool that is included in the Linux distribution. It is used between POSIX-compliant file systems. Traditionally, the snapshot link tree structure provided by rsync with the `--link-dest` option has been considered optimal from a data management perspective because it creates a coherent view of the data set at a given point in time without requiring reference to previous backups. Each directory uses hard links to populate the snapshot with files that are unchanged between jobs.

However, one limitation of rsync is that it builds the file list (that is, traverses the source file tree) before sending data

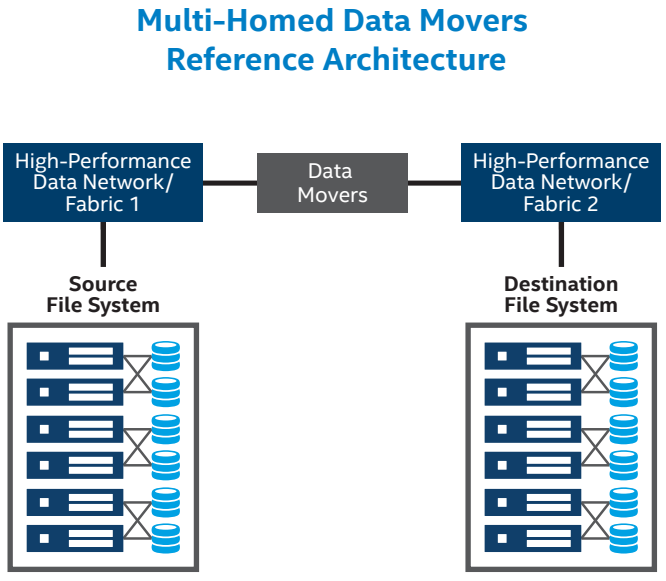

Figure 2. This shows the connectivity between the data movers, source, and destination if the source is on a different network/fabric than the destination.

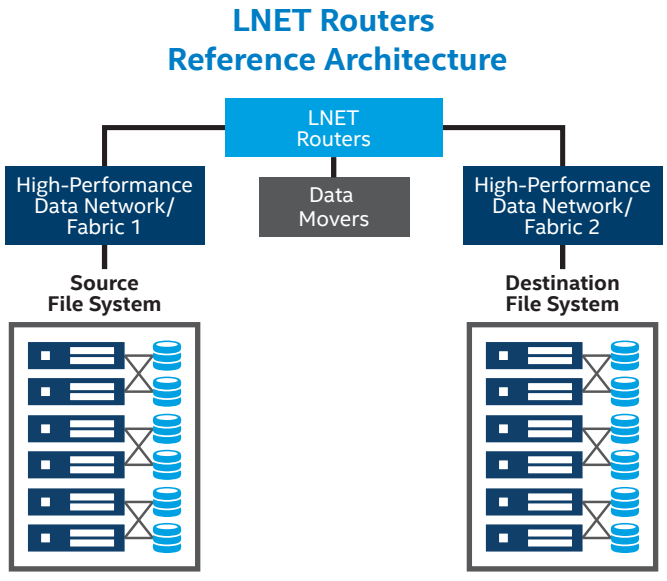

Figure 3. This shows the connectivity between the data movers, source, and destination if the source is on a different network/fabric than the destination. The data movers mount the source file system through the LNET routers.

to the destination file system. The rsync documentation indicates that rsync should start file transfers while it is still building file lists, but rsync does not appear to be particularly effective at sustaining the pipeline.

Fpart is a data migration tool that is based on the hypothesis that if one can start transferring files while the source file tree is still being traversed, one can reduce the elapsed time of the pipeline. The fpart utility is available under BSD license here: https://github.com/martymac/fpart. Fpart is written in C and is distributed as source code. It must be compiled for the target operating system. It has very few runtime dependencies and the 0.9.2 release compiles without any issues on Red Hat Enterprise Linux and CentOS 6.x and 7.x.

Fpart works by walking an arbitrary directory structure and building lists of the tree that it splits into uniform chunks, each of which can be written out to a file. Normally, it builds the lists then writes them out when the walk is complete. However, it also has a "live" mode that generates the lists as they become ready.

Fpart also has hooks that can be executed when the list file is created. The simplest approach is to tell fpart to launch an rsync job with a file list. However, this still results in serialization of the rsync processes, so a little more work is required in order to obtain the desired "live" effect.

Fortunately, in addition to the fpart utility itself, there is a useful wrapper script, written as a UNIX* shell, called `fpsync`. Fpsync acts as a mini job scheduler for running multiple fpart and rsync tasks in parallel across an arbitrary number of nodes. In our evaluation, fpsync was able to distribute multiple tasks in parallel across two data mover clients, but it can also be run on a single node if required.

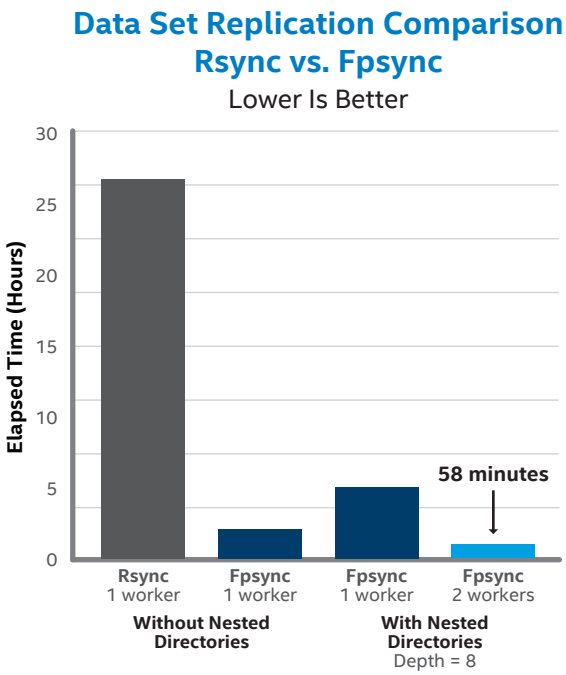

Figure 4. This compares the performance of rsync and fpsync for data set replication between the source and destination.

Figure 4 compares the performance of rsync and fpsync for the replication between the source and destination on a data set of 1 million inodes[4] on each test case.

As shown in Figure 4, fpsync provides considerable performance improvement over rsync. Fpsync scheduled transfers using two data movers, whereas rsync runs on a single data mover.

Fpsync creates a set of jobs to be submitted for each chunk of file lists that fpart generates. A shared directory contains the chunks, and another directory on each data mover node contains a work queue. Fpsync uses the fpart post-hook feature to create a script that it writes to the job queue. A separate function in the fpsync script runs in the background and processes the queue. If there is a new job in the queue it assigns the job to run on one of the data mover nodes, up to the job limit.

By using fpsync, one can distribute file replication work across multiple hosts and run several data transfer processes in parallel. Fpart and fpsync run entirely in the user-space and are file system-agnostic.

The real advantage of combining fpart and fpsync, though, is that there is no state to be kept between runs—each instance of the synchronization execution is self-contained and there is no intermediate data management system or database. Also, one can continue to make use of the snapshot structure that has already been established using the `--link-dest` hard links feature (this needs to be added as an option to fpsync, which is straightforward). There is no need to do any deletes or other manipulation of the backup environment—the file list generated on the source is the definitive catalogue of the live file tree. When transferring to the snapshot site, if the file exists in the previous snapshot and is unchanged it gets a hard link as normal; if it has changed, the changes get transferred; and if the file is new, then it gets sent over in entirety. There should be no need to generate a delete list.

To test fpart and fpsync, a representative data set will be required for testing purposes, installed on an environment that also represents the production storage platform and client nodes used for data migration. Comparison data needs to be collected for execution of rsync, in addition to various configurations of fpart and fpsync. Therefore, a test of rsync against the sample data set should be conducted to provide the runtime baseline.

There is no direct comparison between fpart and rsync, since fpart only creates file lists, but rsync does have a `--dry-run` option that might be comparable. The real goal is to compare fpsync to rsync, but in order to fast-track experimentation, focusing on optimization of fpart initially should help with the tuning of fpsync.

While fpart requires no special configuration to run on a host, the fpsync script relies on passphrase-less Secure Shell (SSH) keys in order to be able to submit jobs to the data movers. Since the software will be executed using the root super-user account, this does represent a security risk.

Fpart Command Syntax

There are several command-line options for fpart and for fpsync (see Tables 3 and 4), so some experimentation is necessary to establish optimum results. The following example provides a reasonable fpart command template:

```
fpart -f <count>  -o <output template> -Z -L
<src dir>
```

Here is a sample fpart command:

```
fpart -f 500  -o /lfs/demo/fp/part.500/o -Z -L
/lfs/demo/src
```

Fpsync Command Syntax

The fpsync command is a bit more complex than the fpart command. The following template is a good starting point:

```
fpsync -n <jobs> -f <nfiles> -w <node>
[-w <node> ...] -d <listdir> <src> <dest>
```

We used the following fpsync command in our evaluation:

```
fpsync -n 160 -f 256 -d <shared-directory>
-w root@<data_mover1> -w root@<data_mover2>
-w root@<data_mover3> -w root@<data_mover4>
-w root@<data_mover5> -w root@<data_mover6>
-w root@<data_mover7> -w root@<data_mover8>
<source> <destination>
```

Fpsync creates directories under /tmp/fpsync providing the logs, partitions, work, and queue information which will help to monitor the jobs progress, scripts, and OUT and ERR files.

Fpart Performance Dashboards

Our evaluation demonstrated that fpart is scalable on both large and small data sets, as shown in Figures 5 and 6 (small files) and Figures 7 through 10 (large files). As mentioned earlier, this performance data is not representative of a production Lustre file system; however, it does provide a baseline comparison. Note that the performance is dependent on multiple factors since the Lustre client must saturate both the read bandwidth at the source and the write bandwidth at the destination.

Table 3. Fpart Command-Line Options

Option	Meaning
`-f`	Limit partitions to <*count*> files.
`-o`	Output partitions to <*output template*>; partitions are sequentially numbered starting with 0.
`-Z`	Treat unreadable directories as empty (implies -z).
`-z`	Pack empty directories (default: pack files only).
`-L`	Live mode: generate partitions during file system crawling.

Table 4. Fpsync Command-Line Options

Option	Meaning
`-n`	Start <*jobs*> concurrent sync jobs. It is recommended to set this number equal to or lesser than the total number of CPU cores on the data movers.
`-f`	Transfer at most <*nfiles*> files per sync job. This number can be up to 256 per partition.
`-w`	Space-separated list of Secure Shell (SSH) workers. For example: `-w 'login@host1 login@host2 login@host3'` or `-w 'login@host1' -w 'login@host2' -w 'login@host3'` Jobs are executed locally if not specified (default).
`-d`	Set the fpsync shared directory to <*listdir*> (absolute path). This option is mandatory when using SSH workers.
`-o`	Override default rsync options with <*options*>. Use this option with care, as certain options are incompatible with a parallel usage (for example, `--delete`).

Figure 5. Fpart read bandwidth at the source with 20 data movers - small files. In our tests, fpart achieved 6.76 GB/s of read bandwidth with 29,000 metadata ops while transferring 100,000 16-MB files at the source file system (results were the same with both 8 and 20 data movers).

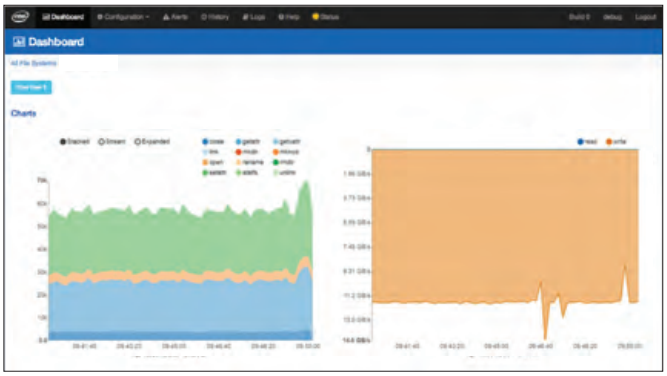

Figure 6. Fpart write bandwidth at destination with 20 data movers - small files. In our tests, fpart achieved 11 GB/s of write bandwidth with 60,000 sustained metadata ops while transferring 100,000 16-MB files at the destination file system (results were the same with both 8 and 20 data movers).

Figure 7. Fpart read bandwidth at source file system with 8 data movers - large files. In our tests, fpart achieved 7.45 GB/s of read bandwidth while transferring one hundred 50-GB files at the source file system with 8 data movers.

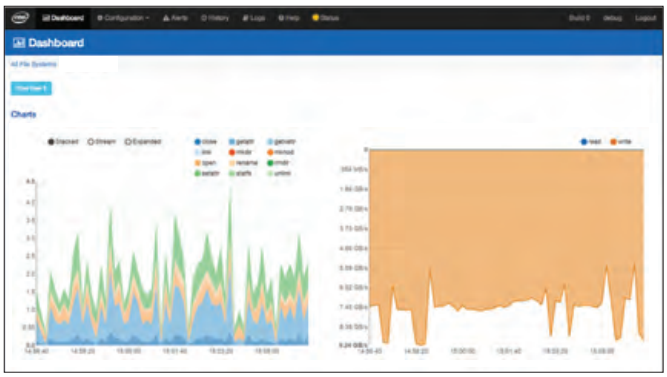

Figure 8. Fpart write bandwidth at destination file system with 8 data movers - large files. In our tests, fpart achieved 7.45 GB/s of write bandwidth while transferring one hundred 50-GB files at the destination file system with 8 data movers.

Figure 9. Fpart read bandwidth at source file system with 20 data movers - large files. In our tests, fpart achieved 8.5 GB/s of read bandwidth while transferring one hundred 50-GB files at the source file system with 20 data movers.

Figure 10. Fpart write bandwidth at source file system with 20 data movers - large files. In our tests, fpart achieved 11 GB/s of write bandwidth while transferring one hundred 50-GB files at the destination file system with 20 data movers.

Fpart Analysis

Our overall analysis of fpart shows advantages and disadvantages.

ADVANTAGES

- It is easy to install and use with few dependencies.
- Multi-threaded, multi-node rsync is efficiently used.
- Logs are available for monitoring.
- It can be used for one-time migration and disaster recovery with scripts managed by cron.

DISADVANTAGES

- It requires minimal documentation.
- It will not split files or stripes; it depends on the backend file system striping for larger files.
- If several paths are provided to fpart, it will examine all of them. If those paths overlap or if the same path is specified more than once, the same files will appear more than once within generated partitions. This is not a bug; fpart does not de-duplicate file system crawling results.

Multi-threaded Copy Program (MCP)

MCP is a high-performance file copy utility that achieves performance gains through parallelization. Multiple files and parts of single files are processed in parallel using multiple threads on multiple processors. The program employs the OpenMP and MPI (Message Passing Interface) programming models. MCP is part of mutil, licensed under the GNU General Public License. We used mutil-1.822.3 for this evaluation, along with coreutils-8.22, both of which are easily compiled on CentOS and Red Hat Enterprise Linux 6.x and 7.x. MCP is available here: https://software.nasa.gov/software/ARC-16494-1.

It is common to use mutil to copy files between local file systems on a daily basis. Files are constantly being moved to locations accessible by systems with different functions and/or storage limits, being backed up and restored, or being moved due to upgraded and/or replaced hardware. Hence, maximizing the performance of copies as well as checksums that ensure the integrity of copies is desirable to minimize the turnaround time of user and administrator activities.

Lustre provides very high performance for such operations using a variety of techniques such as striping files across multiple disks to increase aggregate I/O bandwidth and spreading disks across multiple servers to increase aggregate interconnect bandwidth.

To achieve peak performance from parallel file systems like Lustre, it is typically necessary to utilize multiple concurrent readers/writers from multiple systems to overcome various single-system limitations, such as the number of processors and network bandwidth. The standard cp and md5sum tools of GNU coreutils found on every modern UNIX/Linux system, however, utilize only a single execution thread on a single CPU core of a single system. Therefore, these tools cannot take full advantage of the increased performance of clustered file systems.

Mutil provides the mcp and msum commands, which are drop-in replacements for cp and md5sum that utilize multiple types of parallelism to achieve maximum copy and checksum performance on parallel file systems like Lustre:

- **Multi-threading** is used to keep nodes as busy as possible.
- **Read/write parallelism** allows individual operations of a single copy to be overlapped using asynchronous I/O.
- **Multi-node cooperation** allows different nodes to take part in the same copy/checksum.
- **Split file processing** allows multiple threads to operate concurrently on the same file.
- **Hash trees** allow inherently serial checksums to be performed in parallel.

MCP Syntax

Tables 5 and 6 (on the next page) describe the MCP and msum command-line options. The defaults shown in brackets will vary depending on which compile-time options are used.

Here is the command we used during our evaluation:

```
mpirun --mca plm_rsh_agent "ssh -q -o
StrictHostKeyChecking=no" --allow-
run-as-root -np 400 -npernode 20 -machinefile
hosts / /dm-tools/coreutils-8.22/src/mcp
-pr --stripe-count=1 –T <source> <destination>
```

MCP Performance Dashboards

Our evaluation demonstrated that MCP is more efficient for larger files and needs more optimization for smaller data sets, as shown in Figures 11 through 14 (small files) and Figures 15 through 18 (large files). As mentioned earlier, this performance data is not representative of a production Lustre file system. However, it does provide a baseline comparison. Note that the performance is dependent on multiple factors since the Lustre client must saturate both the read bandwidth at the source and the write bandwidth at the destination.

Figure 11. MCP read bandwidth at source with 8 data movers - small files. In our tests, MCP achieved 2 GB/s of read bandwidth and 30,000 sustained metadata ops while transferring 100,000 16-MB files on the source file system with 8 data movers.

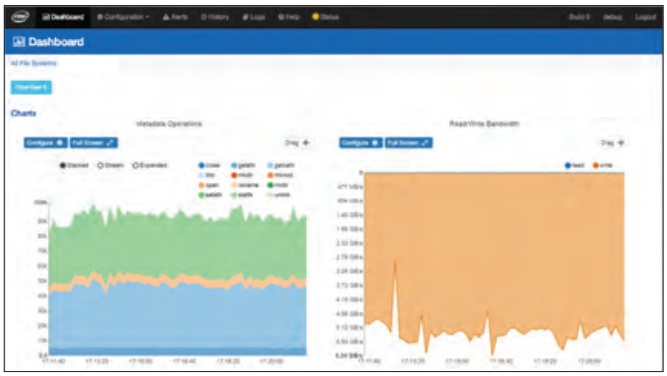

Figure 12. MCP write bandwidth at destination with 8 data movers - small files. In our tests, MCP achieved 6 GB/s of write bandwidth with 100,000 metadata ops while transferring 100,000 16-MB files at the destination file system with 8 data movers. The increase in metadata ops is due to the metadata performance improvements on Intel® EE for Lustre* 3.0 with ZFS as the backend file system.

Figure 13. MCP read bandwidth at source with 20 data movers - small files. In our tests, MCP achieved 6.76 GB/s of read bandwidth with 29,000 sustained metadata ops while transferring 100,000 16-MB files on the source file system with 20 data movers.

Figure 14. MCP write bandwidth at destination with 20 data movers - small files. In our tests, MCP achieved 11 GB/s of write bandwidth and 60,000 sustained metadata ops while transferring 100,000 16-MB files at the destination file system with 20 data movers.

Table 5. MCP Command-Line Options

Option	Meaning
`--buffer-size=<mbytes>`	Read/write buffer size [4]
`--check-tree`	Print hash subtrees to pinpoint corruption
`--direct-read`	Enable use of direct I/O for reads
`--direct-write`	Enable use of direct I/O for writes
`--double-buffer`	Enable use of double buffering during file I/O
`--dst-offset=<pos>`	Copy to destination file beginning at <*pos*>
`--fadvise-read`	Enable use of posix_fadvise during reads
`--fadvise-write`	Enable use of posix_fadvise during writes
`--hash-leaf-size=<kbytes>`	Granularity of hash tree [1048576]
`--hash-type=<type>`	Hash type [MD5]. <*type*> can be one of the following: `md5 sha1 sha256 sha384 sha512 sha224 crc32 crc32rfc1510 crc24rfc2440`
`--length=<len>`	Copy <*len*> bytes beginning at `--offset` (or from 0 if `--offset` is not specified)
`--listen-port=<port>`	Listen on <*port*> for requests from cooperating hosts
`--manager-host=<host>`	Host name or IP address of the management thread for multi-node/multi-host copies
`--manager-port=<port>`	Port on which to contact the management thread
`--mpi`	Enable use of Message Passing Interface (MPI) for multi-node copies
`--offset=<pos>`	Copy `--length` bytes beginning at <*pos*> (or to end if `--length` is not specified)
`--password-file=<file>`	File to use for passwords (will be created if it does not exist)
`--print-hash`	Print hash of each file to stdout similar to md5sum, with sum of the src file computed, but dst file name printed so that md5sum `-c` can be used on the output to check that the data written to disk was what was read
`--print-src`	Print src instead of dst in `--print-hash` and `--print-stats}`
`--print-stats`	Print performance per file to stderr
`--print-stripe`	Print striping changes to stderr
`--read-stdin`	Perform a batch of operations read over stdin in the form 'SRC DST RANGES' where SRC and DST must be URI-escaped (RFC 3986) file names and RANGES is zero or more comma-separated ranges of the form 'START-END' for 0 <= START < END
`--skip-chmod`	Retain temporary permissions used during copy
`--split-size=<mbytes>`	Size to split files for parallelization [1024]
`--stripe-count=<count>`	Absolute number of stripes. When followed by 's' (as in 1s, 2s, 3s) it specifics stripes per src GBs. When followed by 'l' it specifies stripes per `--length` GBs.
`--threads=<number>`	Number of OpenMP worker threads to use [4]

Table 6. Msum Command-Line Options

Option	Meaning
`--buffer-size=<mbytes>`	Read/write buffer size [4]
`--check-tree`	Print/check hash subtrees to pinpoint corruption
`--direct-read`	Enable use of direct I/O for reads
`--double-buffer`	Enable use of double buffering during file I/O
`--fadvise-read`	Enable use of posix_fadvise during reads
`--hash-leaf-size=<kbytes>`	Granularity of hash tree [1048576]
`--hash-type=<type>`	Hash type [MD5]. <*type*> can be one of the following: md5 sha1 sha256 sha384 sha512 sha224 crc32 crc32rfc1510 crc24rfc2440
`--length=<len>`	Hash <*len*> bytes beginning at `--offset` (or 0 if `--offset` is not specified)
`--listen-port=<port>`	Listen on <*port*> for requests from cooperating hosts
`--manager-host=<host>`	Host name or IP address of the management thread for multi-node/multi-host copies
`--manager-port=<port>`	Port on which to contact the management thread
`--mpi`	Enable use of Message Passing Interface (MPI) for multi-node checksums
`--offset=<pos>`	Hash `--length` bytes beginning at <*pos*> (or to end if `--length` is not specified)
`--password-file=<file>`	File to use for passwords (will be created if it does not exist)
`--read-stdin`	Perform a batch of operations read over stdin in the form 'FILE RANGES' where FILE must be a URI-escaped (RFC 3986) file name and RANGES is zero or more comma-separated ranges of the form 'START-END' for 0 <= START < END
`--split-size=<mbytes>`	Size to split files for parallelization [0]
`--threads=<number>`	Number of OpenMP worker threads to use [4]

Figure 15. MCP read bandwidth at source with 8 data movers - large files. In our tests, MCP achieved 8 GB/s of read bandwidth while transferring one hundred 50-GB files at the source file system with 8 data movers.

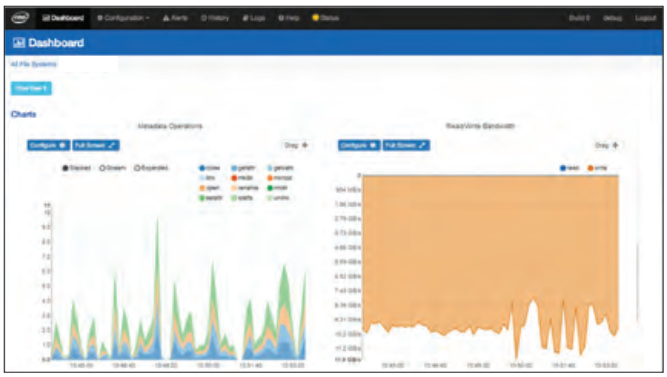

Figure 16. MCP write bandwidth at destination with 8 data movers - large files. In our tests, MCP achieved 10 GB/s of write bandwidth while transferring one hundred 50-GB files at the destination file system with 8 data movers.

Figure 17. MCP read bandwidth at source with 20 data movers - large files. In our tests, MCP achieved 8.5 GB/s of read bandwidth while transferring one hundred 50-GB files at the source file system with 20 data movers.

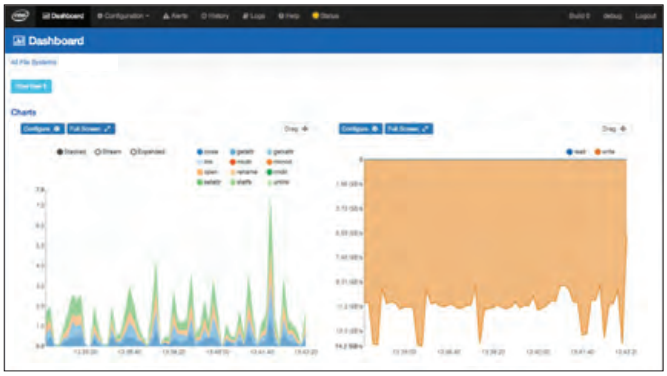

Figure 18. MCP write bandwidth at source with 20 data movers – large files. In our tests, MCP achieved 11 GB/s of write bandwidth while transferring one hundred 50-GB files at the destination file system with 20 data movers.

MCP Analysis

Our overall analysis of MCP shows advantages and disadvantages.

ADVANTAGES

- It is easy to use and install.
- It is MPI-driven and has an option to provide the stripe size.
- It is good for larger files.
- Msum (parallel checksum) is an added advantage.
- It can be used for both one-time migration and live migration/disaster recovery.

DISADVANTAGES

- It requires minimal documentation.
- Small file performance needs to be optimized.
- Logging facility is not available.

Future Work

In addition to the data migration best practices, recommendations, and tools and techniques covered in this document, there is still additional work that needs to be done to enable a seamless migration:

- Development of scripts that require minimal management
- Cron configuration for both fpart and MCP

The following data migration tools are worthy of investigation, as they are completely enabled and managed by a distributed queue system (which needs to be evaluated in the future).

Lustre data mover from San Diego Super Computer Center (https://github.com/sdsc/lustre-data-mover). This tool is built around RabbitMQ* and Python Celery* scripts. The advantages of this tool include management of data movers with scripts, centralized logging, and performance monitoring. Workers can be configured through the HPC cluster compute nodes and the worker's pool configuration

looks impressive. This could be a good solution for live migration of data or disaster recovery, since the migration happens based on atime and mtime.

Psync data mover (https://github.com/ncsa/psync) is another option to investigate, since it is enabled with parallel rsync at both the directory and file level. Psync is managed by Python Celery scripts, RabbitMQ, and Redis* with centralized logging. Because it is enabled with parallel rsync, it may be a good fit for both small and large data sets (based on our experience with fpart). Psync could be another solution for live migration of data or disaster recovery, since the migration happens based on atime and mtime.

Conclusion

The information in this document can help system integrators and administrators plan and define the best possible implementation of data migration using fpart and MCP. Careful planning, an informed architecture choice, and the appropriate use of data migration tools can help make data migration go smoothly and efficiently, with minimal disruption to day-to-day processes. Whether the source is a Lustre file system or some other POSIX-compliant file system, migrating data to Intel EE for Lustre software will help future-proof the enterprise HPC storage infrastructure by enabling the enterprise to easily keep pace with an ever-growing flood of data.

For more information visit **intel.com/lustre** or email **hpdd-sales@intel.com.**

Learn More

You may find the following resources useful:

- **fpart:** sourceforge.net/projects/fpart
- **mutil:** people.nas.nasa.gov/~kolano/projects/mutil.html
- **Lustre Data Mover presentation:** http://cdn.opensfs.org/wp-content/uploads/2016/04/LUG2016D3_Lustre-Data-Mover_Wagner.pdf
- **Parallel Synchronization of Multi-Pebibyte File Systems presentation:** http://lustre.ornl.gov/ecosystem/documents/tech/Loftus-NCSA-Psync.pdf

[1] Discussion of any open source, third-party tool does not imply endorsement of that tool by Intel, nor does it imply any responsibility for supporting that tool.

[2] ZFS is a combined file system and logical volume manager designed by Sun Microsystems.

[3] Community ENTerprise Operating System. A free rebuild of source packages from the Red Hat Enterprise Linux.

[4] An inode is a data structure used to represent a filesystem object.

0816/EYAR/KC/PDF ♻ Please Recycle 334704-001