I/O Performance Characterization of Lustre and NASA Applications on Pleiades

Subhash Saini, Jason Rappleye, Johnny Chang, David Barker, Piyush Mehrotra, Rupak Biswas
NASA Advanced Supercomputing (NAS) Division
NASA Ames Research Center
Moffett Field, California 94035-1000, USA
{subhash.saini, jason.rappleye, johnny.chang, david.p.barker, piyush.mehrotra, rupak.biswas}@nasa.gov

Abstract—In this paper we study the performance of the Lustre file system using five scientific and engineering applications representative of NASA workload on large-scale supercomputing systems such as NASA's Pleiades. In order to facilitate the collection of Lustre performance metrics, we have developed a software tool that exports a wide variety of client and server-side metrics using SGI's Performance Co-Pilot (PCP), and generates a human readable report on key metrics at the end of a batch job. These performance metrics are (a) amount of data read and written, (b) number of files opened and closed, and (c) remote procedure call (RPC) size distribution (4 KB to 1024 KB, in powers of 2) for I/O operations. RPC size distribution measures the efficiency of the Lustre client and can pinpoint problems such as small write sizes, disk fragmentation, etc. These extracted statistics are useful in determining the I/O pattern of the application and can assist in identifying possible improvements for users' applications. Information on the number of file operations enables a scientist to optimize the I/O performance of their applications. Amount of I/O data helps users choose the optimal stripe size and stripe count to enhance I/O performance. In this paper, we demonstrate the usefulness of this tool on Pleiades for five production quality NASA scientific and engineering applications. We compare the latency of read and write operations under Lustre to that with NFS by tracing system calls and signals. We also investigate the read and write policies and study the effect of page cache size on I/O operations. We examine the performance impact of Lustre stripe size and stripe count along with performance evaluation of file per process and single shared file accessed by all the processes for NASA workload using parameterized IOR benchmark.

Key words: Lustre file system, I/O performance evaluation, benchmarking, computational fluid dynamics, climate modeling, Read and Write Policy, I/O cache effect, I/O latency.

I. INTRODUCTION

Several scientific and engineering applications running on petaflop class supercomputers deal with increasingly large data sets, and thus, the time required for input and output of data can become a significant bottleneck [1]. It is important for supercomputers to be not only balanced with respect to the compute processor, memory, and interconnect, but also with respect to the I/O performance. That is, it is not just the number of petaflops per second that matters, but also how many gigabytes per second or terabytes per second of data can applications really move in and out of disks that will affect whether these high performance computing systems can be used productively for new scientific discoveries. It is important that application scientists begin to examine the characteristics of the I/O resources available to them and how to best utilize their capabilities. Parallel file systems such as Lustre [2-4] are becoming very large, especially when supporting petaflop class systems such as NASA's Pleiades system. In order to address these issues, the performance, stability, robustness, and reliability of the Lustre parallel file system needs to be studied.

In addition, recently the Open Scalable File Systems (OpenSFS) Benchmarking Workgroup has been formed with a plan to provide an I/O benchmark suite for the scalable parallel file system administrators and users of petaflop class computing facilities [5]. NASA is a member of OpenSFS and its benchmarking working group. As a first step, this group aims to characterize the I/O workloads of parallel file systems deployed at various high-performance computing facilities. Using these characteristics, the working group will develop a suite of I/O benchmarks to emulate these workloads.

Recently, several researchers have conducted performance evaluation and characterization of parallel file systems such as CXFS, GPFS, PVFS2, Lustre, etc. Saini et al used I/O benchmarks and applications on SGI Altix and NEC SX-8 super clusters [6]. Using the MADbench2 benchmark Borrill et al studied the I/O performance on several supercomputers ([7-8]. Yu et al characterized the performance of several I/O benchmarks on the Lustre file system [9]. These investigations did not collect performance metrics of Lustre file system using performance monitors or measured the overhead of I/O operations using system calls and signals.

To the best of our knowledge, our contributions in this paper are as follows:

- Conducted a survey of the NASA scientific and engineering workload applications to characterize the I/O requirements and thus define the parameters for I/O benchmarks.
- Developed the NAS Lustre Performance (NLP) package to collect Lustre performance data at the end of a batch job.
- Used our newly developed performance tool to collect Lustre performance data for five production quality NASA applications from different disciplines such as structured and unstructured computational fluid dynamics, climate and cosmology [12-16].
- Investigated the latency of read and write system calls under Lustre and NFS on Pleiades [2-4, 17-18]

- Investigated the read and write policies of Lustre.
- Investigated the effect of page cache on I/O.
- Investigated the optimal Lustre stripe size and stripe count on Pleiades using I/O parameters representative of NASA workload.
- Modeled and parameterized the NASA workload using IOR benchmark [19] to study the performance of file per process and single shared file paradigms.

The remainder of the paper is organized as follows. Section II gives details of the NASA I/O workload. In section III we present the computing platform and I/O file system used in this paper. Section IV presents performance metrics of the Lustre parallel file system. Section V describes the I/O benchmarks used in the present study. Section VI presents the methodology followed in the paper. Section VII gives an overview of production level applications used. Section VIII gives the results of our investigation. Finally, Section IX contains the conclusion and future work.

II. NASA I/O Workload

Based on our survey of the NASA scientific and engineering applications, we characterized the I/O requirements of typical applications run on our system based on the following parameters:

- access pattern (random/sequential and read/write),
- size of each read and write operation,
- file type (shared: all processes read/write one shared file, or one-file-per-processor: each process reads/writes its own separate file), and
- programming interface (POSIX, MPI-IO, HDF5, Parallel-HDF5, NetCDF and pNetCDF).

The major results of this survey include:

- Random access is rare for NASA applications; I/O access is dominated by sequential operations.
- Write is dominated by append-only writes
- I/O read and write sizes vary widely: from a few KB to several GB.
- The majority of applications perform sequential I/O where each process/rank sends its data to the master (rank 0), which then writes the data to a single file. The advantages of this approach are that it is simple and the performance is reasonable for small IO sizes. The disadvantages are that it is not scalable and efficient, slow for large number of processes (ranks) or data sizes, and may not be possible if rank 0 is memory constrained.
- Few applications use HDF5 and NetCDF.

Five production-quality NASA applications were selected that provide good coverage of the above set of I/O characteristics [12-16]. They are described in Section VII.

III. Computing Platform and I/O File System

NASA's Pleiades supercomputer system is located at NASA Ames Research Center. Pleiades comprises 11,776 nodes (126,720 cores) based on four different Intel Xeon processors: Harpertown, Nehalem-EP, Westmere-EP and Sandy Bridge-EP. The nodes are interconnected with three generations of InfiniBand (IB) network in a hypercube topology: DDR, QDR and FDR data rates. In this study, we used only the Westmere-EP based nodes using QDR IB interconnect [18]. Pleiades has two file systems, namely, an NFS home file system and a Lustre parallel file system.

A. *Home File System: NFS*

The home file system on Pleiades is exported from an SGI XE500 with two quad-core Nehalem processors and 48 GB of RAM. It is a Network File System (NFS) mounted on all of the Pleiades front-ends, bridge nodes and compute nodes [17]. It consists of a single 4 + 1 RAID 5 volume on an SGI IS220 controller, providing 1 TB of usable storage [18].

B. *Parallel File System: Lustre*

The Lustre file system is composed of four components: Lustre clients, object storage servers (OSS), object storage targets (OST), and Metadata servers (MDS). Figure 1 is a schematic diagram of these four components of Lustre.

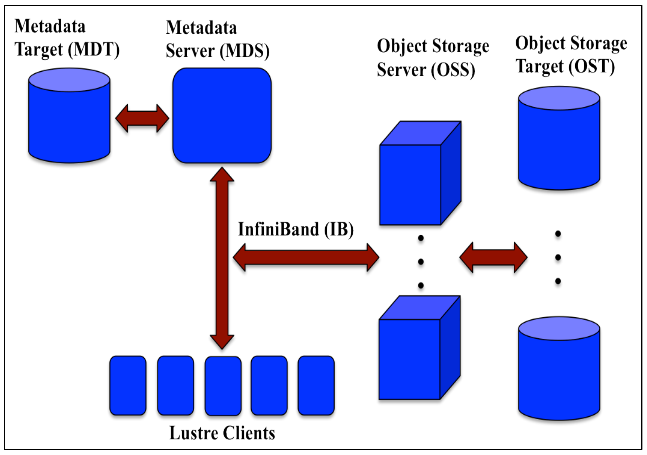

Figure 1. Lustre components.

The file metadata is controlled by a MDS and stored on a Metadata Target (MDT). OSSs manage a set of OSTs by controlling I/O access and handling network requests to them. OSSs contain metadata about the files stored on their OSTs. OSTs are block storage devices that store user file data in one or more objects, with each object stored on a separate OST.

Pleiades has six Lustre file systems each containing one MDS and one MDT, eight OSSs, 60 to 120 OSTs that provide a total of 6.8 PB of storage and serve thousands of compute nodes. MDT sizes range from 0.6 TB to 0.9 TB [18]. Sizes of OSTs are from 7.1 TB to 15 TB. Total space available in each file system is from 424 TB to 1.7 PB. The default stripe size and stripe count are 4 MB and 1, respectively. Currently, Lustre version 1.8.6 is used to manage these file systems.

IV. Lustre Performance Metrics

Lustre provides a wealth of performance information on both clients and servers via the Linux *proc* file system. This includes the client read and write remote procedure call (RPC) size distribution, metadata operation counters, distributed lock manager metrics, and the per block-device I/O sizes. This performance data can be used to characterize the performance of one or more Lustre clients in aggregate—usually in terms of a single batch job—and also of the Lustre servers themselves.

The bulk of the analysis performed in this paper is the result of examining the RPC size distributions and metadata operation counts when running various applications on Pleiades. Lustre, being a POSIX-compliant file system, presents a unified file system interface such as *open(), read(), write(),* etc. to the user. In Linux, this unified interface is achieved through the Virtual File System (VFS) layer. There is a thin layer in Lustre called Lustre Lite (*llite*) that is hooked with VFS to present that interface. The file operation requests that reach *llite* go through the whole Lustre software stack to access the Lustre file system.

It is worth noting that for many reasons, there is not a one-to-one correspondence between system calls and RPCs, and also between RPCs and disk I/O. For example, when buffered I/O is used (as opposed to direct I/O), applications write system calls result in dirtying pages in the page cache. Lustre will aggregate multiple pages together when sending an RPC. Thus, a series of small sequential writes may result in a much larger RPC being sent to the server. For read system calls to sequential locations in a file, the Lustre read-ahead mechanism can result in a larger read RPC being issued than the size specified in the system call itself.

V. I/O BENCHMARKS

In this section we describe the I/O benchmarks used in our study.

A. *Sequential I/O Benchmark*

Sequential Write Read (SWR) is a single process I/O benchmark that writes and reads any size of file using various block sizes, stripe sizes and stripe counts. This benchmark mimics sequential I/O where all the processes send data to a master process, which writes to disk.

B. *IOR HPC Benchmark:*

Lawrence Livermore National Laboratory (LLNL) developed Interleaved Or Random (IOR) benchmark to procure their supercomputers [19]. It can do parallel/sequential read/write operations that are typical in scientific applications. It has options for one file per process or single shared file accesses by all the processes. Furthermore, its API provides the option for modern file systems such as POSIX (shared or unshared), MPI-IO, pHDF5, and pNetCDF.

VI. Methodology

In this section we describe the methodology adopted in this study. All I/O runs were done during production time so performance depends on other jobs running on the system. We ran each benchmark five times and present the best value.

A. *Package for Collecting Lustre Performance Metrics*

We have developed the NAS Lustre Performance (NLP) package, based on two components of SGI's PCP package [10], to collect Lustre performance data and produce human-readable reports at the end of each Pleiades batch job. The package consists of two main components. First is an agent that uses the Performance Metrics Domain Agent (PMDA) interface [11] to collect performance metrics provided by Lustre in the *proc* file system. In general, a PMDA collects a specific set of metrics and implements a specific set of API calls that are used by another daemon to fetch the data when it is needed.

As the second component of the NLP package, we have implemented a set of scripts that gathers metrics from the agent, aggregates them together, and generates the Lustre performance report for each job. The Lustre metrics are collected from each compute node in a job using a script that invokes the performance metrics value dumper (pmval) command [20] provided in the Performance Co-Pilot (PCP) package. The pmval command retrieves the value of a metric from a local or remote host. The metadata, bytes read, bytes written, and RPC histogram are collected and stored on a per file system basis. The RPC histograms require some additional processing. The Lustre clients store the histograms on a per-OST basis, therefore the RPC counts must be added up for all OSTs on each file system. The metrics are collected at the beginning and end of each job. The delta is calculated, and a report is generated that is included in the output for the user's job.

Sample performance statistics for the Enzo application reads and writes to Lustre file system extracted by the NLP package is shown in Tables I and II. The statistics block lists the number of Lustre operations and the volume of Lustre I/O generated for each file system. The I/O volume is listed in total, and is broken out by RPC size. The following metadata operations statistics are also listed:

- Number of file opens and closures on the Lustre file system
- Number of *stat* and *statfs* query operations invoked by commands such as *"ls -l"* and *"du"*
- Total amount of data read and written in gigabytes.

TABLE I. EXTRACTED PERFORMANCE METRICS USING NLP PACKAGE.

I/O	**RPC Size (KB)**								
	4	8	16	32	64	128	256	512	1024
# Read	12243	264	56	81	242	269	543	1812	26572
# Write	393	62	198	155	96	74	199	595	52606

TABLE II. EXTRACTED PERFORMANCE METRICS USING NLP PACKAGE.

Number of files		stats		Amount of data (GB)	
Open	**Close**	**stat**	**statfs**	**Read**	**Write**
5303	5769	12460	0	54	56

The read and write operations are further broken down into buckets based on RPC size. In Table I, the first bucket reveals that 12243 data reads occurred in blocks between 0 and 4 KB in size, 264 data reads occurred with RPC sizes between 4 KB and 8 KB, and so on. As noted before, the RPC size data may be affected by library and system operations and, therefore, could differ from expected values. That is, small reads or writes by the program might be aggregated into larger RPC operations, and large reads or writes might be broken into smaller pieces. High counts in the smaller buckets in the I/O pattern of the application are an indication of I/O inefficiency.

These client-side metrics can also be useful in detecting problems after a Lustre upgrade. For example, a recent regression in the read ahead code caused some previously well-behaved access patterns - resulting in mostly 1 MB RPCs – to generate 8 KB RPCs instead. Comparison of per-job Lustre metrics for similar workloads, before and after an upgrade, can help to uncover future regressions in the Lustre file system client code.

B. *IOR Parameters*

The IOR benchmark provides an option of choosing several parameters [19]. Table III gives IOR parameters we used in our study. With appropriate choice of the IOR parameters, one can emulate I/O pattern to closely match the data access pattern of applications.

TABLE III. IMPORTANT IOR BENCHMARK PARAMETERS.

Parameter	Description	Parameter Choices
API	File format	POSIX, MPI-IO, HDF5
FilePerProc	One/file/proc, shared	True or False
WriteFile	Write file on disk	True or False
ReadFile	Read file from disk	True or False
NumTasks	Number of tasks	System limited
BlockSize	Blocks to write/task	KB, MB or GB
TransferSize	I/O transaction/task	Divisible by BlockSize

C. *Lustre Parameters: Stripe Size and Stripe Counts*

A key feature of the Lustre file system is its capability to distribute the pieces of a file across several OSTs, essentially a set of parallel IO disks, using a technique called file striping.

A file is striped when data is separated into stripes (small chunks), so that read and write operations can access multiple OSTs concurrently. Stripe size is the amount of data to store on one OST before moving to the next. Stripe count is the number of OSTs over which to stripe a file.

File striping will most likely improve performance of applications that read or write to a single or multiple large shared files. Striping will likely have little effect for the following types of I/O patterns:

- Sequential I/O where a single process performs all the I/O, (stripe size will have little effect, but stripe count does have a large effect).
- Multiple nodes perform I/O, but access files at different times.
- Multiple nodes perform I/O simultaneously to different files that are small (each < 100 MB).

Storing a single file across multiple OSTs may increase the bandwidth available when accessing the file. However, striping has disadvantages, namely, increased overhead due to network operations and having to access multiple servers.

VII. Science and Engineering Applications

We used the following five production quality full applications representative of NASA's workload to collect the Lustre performance metrics on Pleiades. Brief description of these applications is given below.

OVERFLOW-2 is a general-purpose Navier-Stokes solver for CFD problems [12]. The code uses finite differences in space with implicit time stepping. It uses overset-structured grids to accommodate arbitrarily complex moving geometries. The dataset used is a wing-body-nacelle-pylon geometry (DLRF6), with 23 zones and 36 million grid points. The input dataset is 1.6 GB in size, and the solution file is 2 GB.

CART3D is a high fidelity, inviscid CFD application that solves the Euler equations of fluid dynamics [13]. It includes a solver called Flowchart, which uses a second-order, cell-centered, finite volume upwind spatial discretization scheme, in conjunction with a multi-grid accelerated Runge-Kutta method for steady-state cases. In this study, we used the geometry of the Space Shuttle Launch Vehicle (SSLV) for the simulations. The SSLV uses 24 million cells for computation, and the input dataset is 1.8 GB and output file is 1 GB. The application requires 16 GB of memory to run.

USM3D is a 3-D unstructured tetrahedral, cell-centered, finite volume Euler and Navier-Stokes flow solver [14]. Spatial discretization is accomplished using an analytical reconstruction process for computing solution gradients within tetrahedral cells. The solution is advanced in time to a steady-state condition by an implicit Euler time-stepping scheme. The test case used 10 million tetrahedral meshes, requiring about 16 GB of memory and 10 GB of disk space. Input and output files are 1 GB and 8 GB respectively.

MITgcm (MIT General Circulation Model) is a global ocean simulation model for solving the fluid equations of motion using the hydrostatic approximation [15]. The MITgcm test case uses 50 million grid points and requires 32 GB of system memory and 20 GB of disk to run. Input file is 1 GB. It writes checkpoint file of 8 GB of data using Fortran I/O. The test case is a ¼ degree global ocean simulation with a simulated elapsed time of two days.

Enzo is an adaptive mesh refinement (AMR), grid-based hybrid parallel code for astrophysics and cosmology simulations and uses hybrid physics (fluid + particle + gravity + radiation) and has physics capabilities like ideal magneto hydro dynamics (MHD), radiation transport (ray tracing and flux limited diffusion), star particle class, metallicity-dependent cooling, and several new hydro solvers [16]. Input and output files are 54 GB and 56 GB respectively. The root grid is read into the root core and then partitioned to separate cores using MPI communication.

VIII. Results

In this section we present the performance metrics (amount of read and write, number of file opens and closures and RPC size distribution) for five NASA applications extracted by the NLP package we developed. We also present the latency of open and close operations by monitoring all the relevant system calls using a Linux utility called *Strace* for both NFS and Lustre file system. Finally, we characterize the Pleiades Lustre file system to determine the optimal stripe size and stripe counts that can enhance the performance of the applications.

A. Extraction of Performance Metrics

In this section we present the performance metrics such as total amount of data read and written, total number of file opens and closures and RPC size distribution of write and read data.

1) Amount of Read and Write Data

Figure 2 shows total amount of data read and written by the five applications for our chosen datasets. Amount of read data is 1 GB, 2 GB, 4 GB, 6 GB, and 54 GB for MITgcm, Overflow, Cart3D, USM3D and Enzo respectively. The smallest grid read is by MITgcm, whereas Enzo reads the largest. All five applications perform sequential I/O where the master process reads the input data and then either broadcasts (Cart3D) or uses sends/receives to communicate the relevant portions to the other processes. Amount of write data is 9 GB, 3 GB, 1 GB, 1 GB, and 56 GB for MITgcm, Overflow, Cart3D, USM3D and Enzo respectively. For write, the master process collects data from other processes and then writes to the file. For applications with large grid files or large output files (i.e. checkpoint, restart, or visualization files) sequential I/O is a bottleneck, especially with large numbers of cores. Although these applications perform sequential I/O, they can benefit from using large stripe size and stripe counts as discussed in Section VIII-C-2.

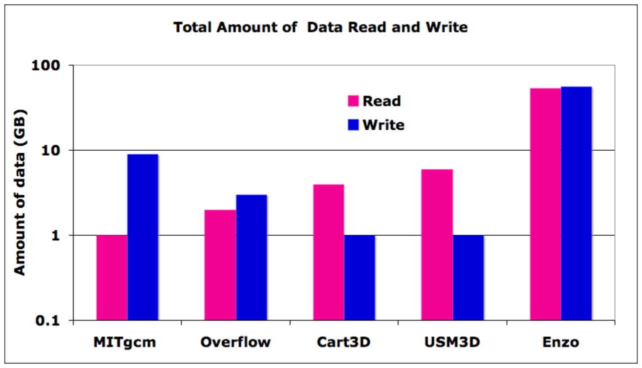

Figure 2. Amount of read and write data for five applications.

2) Number of File Opens and Closes

Figure 3 shows the number of files opened and closed by three applications (USM3D, Overflow and Cart3D) for cores ranging from 32 to 128. For all three applications, the number of file opens and closures are under 60 and increase with increasing number of cores as each core writes its own intermediate data during computation. However, major potions of the I/O are done while reading in a grid file at the beginning and writing a checkpoint or restart file at the end. Figure 4 shows the corresponding data for MITgcm and Enzo. Number of file opens and closures for these two applications is much higher than those of Figure 3. It is clear that MITgcm and Enzo are I/O intensive and will benefit from using optimal stripe size and stripe count (see Section VIII-C-2). Large numbers of file opens and closures in MITgcm and Enzo lead to poor scalability as overhead (latency) in open, close and read/write in Lustre is very high (see Section VIII-B).

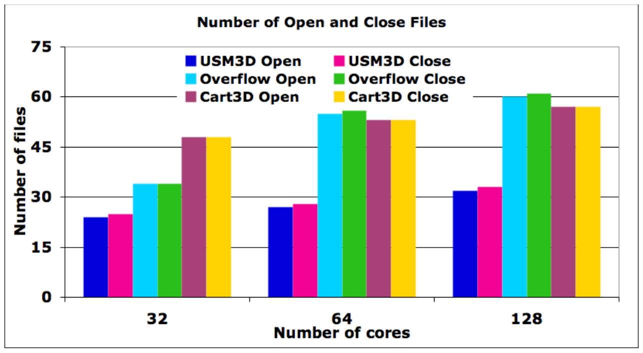

Figure 3. Number of files opens and closes for three applications.

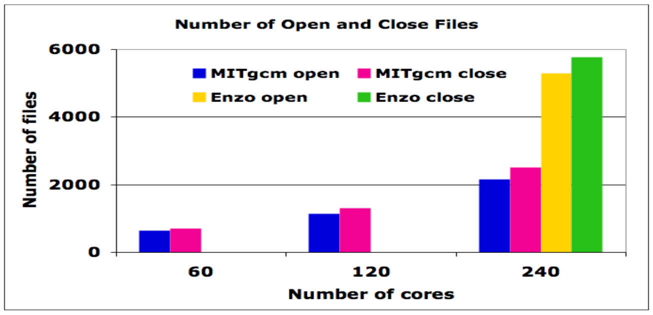

Figure 4. Number of file opens and closes for MITgcm and Enzo.

3) *RPC Size Distribution:*

In this subsection we present RPC size distribution of read and write as measured by our newly developed tool for five applications (Overflow, Cart3D, USM3D, MITgcm and Enzo) under Lustre on Pleiades.

Overflow: Figure 5 shows RPC size distribution under Lustre for Overflow on number of cores ranging from 8 to 128. Most of the reads and writes (number of RPC blocks ranging from 1550 to 1950) are with RPC size of 1024 KB on all the cores. Number of reads (input grid file of 2 GB) with RPC size 4 KB increase gradually from 264 to 2016 with increasing core counts from 8 to 128 whereas corresponding writes (3 GB restart/output file) remain almost constant between 22 and 27. Number of reads is higher than writes by a factor of 10 and 20 for 256 KB and 512 KB RPC sizes. However, for an RPC size of 1024 KB, number of writes is higher than reads by 400. Large number of RPC for 4 KB is an indication of inefficiency of Lustre file system.

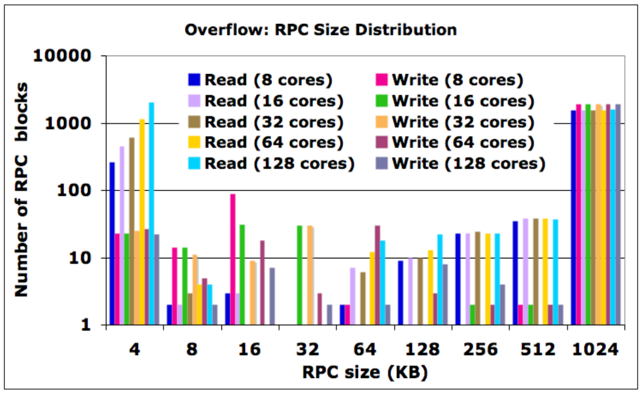

Figure 5. RPC size distribution under Lustre for Overflow.

Cart3D: Figure 6 shows RPC size distribution under Lustre for Cart3D on number of cores ranging from 8 to 128. Most of the reads (input grid file is 4 GB) and writes (output file of 1 GB) are either for RPC sizes 4 KB or 1024 KB. For an RPC size of 4 KB, number of reads are always much higher than number of writes and increase with core counts. However, for an RPC size of 1024 KB, number of reads is almost constant ranging from 1662 to 1672 and there are no writes.

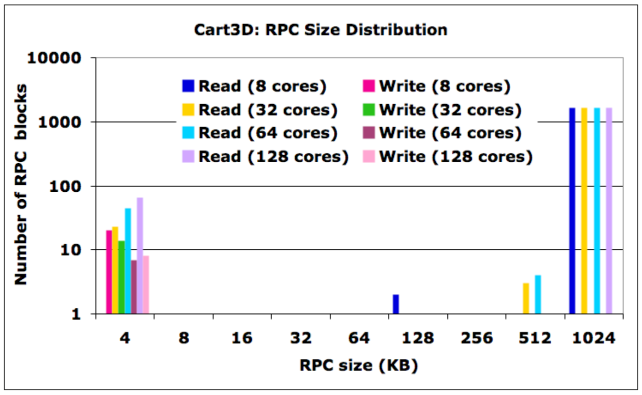

Figure 6. RPC distribution under Lustre for Cart3D.

USM3D: Figure 7 shows RPC size distribution for USM3D for 32, 64 and 128 cores. Most of the writes (1 GB) are done using RPC size of 1024 KB. However, reads (6 GB grid file) are done using RPC sizes of 4 KB and 1024 KB.

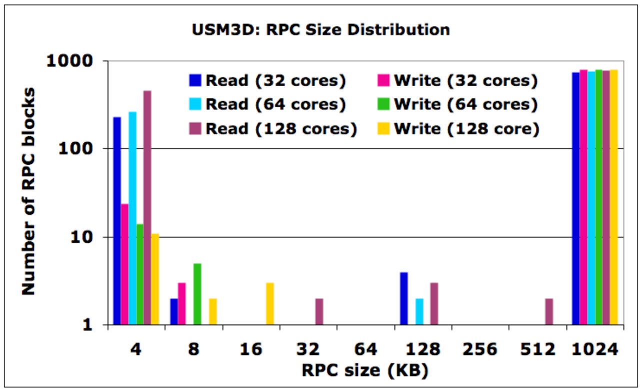

Figure 7. RPC size distribution under Lustre for USM3D.

MITgcm: Figure 8 shows RPC size distribution for MITgcm for cores ranging from 64 to 240. Most reading (grid file of 1 GB) is done using RPC size 4 KB whereas most of the writes (check-point file of 8 GB) use 1024 KB RPC size. In addition to the checkpoint file, MITgcm does a lot of other writes amounting to 1 GB as is evident from the very the high numbers of file opens and closures (see Figure 4) and uses RPC sizes ranging from 4 KB to 1024 KB. Most of the writes (8 GB out of total of 9 GB) are a final checkpoint file and a remaining 1 GB is written by thousands of cores, which need to perform I/O and these small I/O operations (including open, close, read, write, etc.) are very expensive (see Section VIII-B). This performance bottleneck in MITgcm related to opening and closing thousands of files has been detected for the first time by our Lustre performance metric extraction tool.

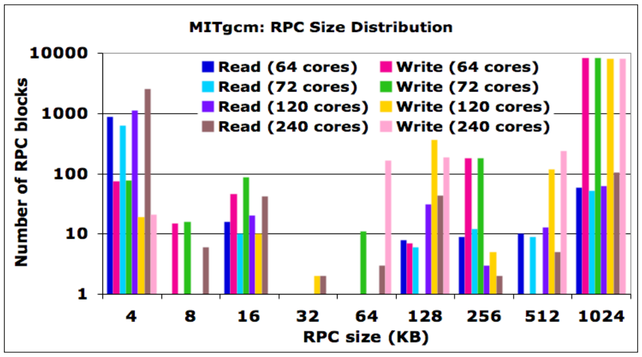

Figure 8. RPC size distribution under Lustre for MITgcm

Enzo: Figure 9 shows RPC size distribution of Enzo for RPC sizes ranging from 4 KB to 1024 KB. Amount of read and write data is 54 GB and 56 GB respectively. Most of the reads and writes are done using an RPC size of 4KB and 1024KB. For 4 KB, the number of reads and writes are 12243 and 393 respectively and corresponding numbers for 1024 KB are 36576 and 52606.

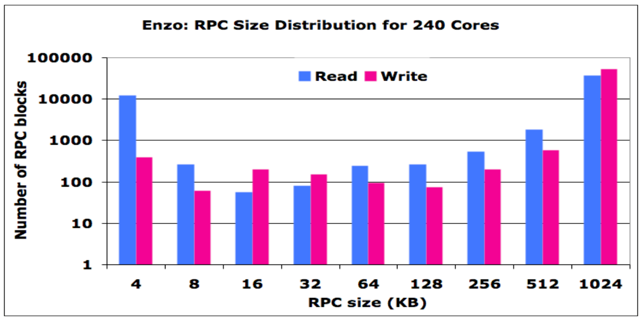

Figure 9. RPC size distribution under Lustre for Enzo.

In summary, large number of RPC for 4 KB is an indication of inefficiency of Lustre file system because latency for 4 KB RPC size is 256 times higher than 1024 KB RPC size.

B. *Lustre Read and write Latency*

The performance indicator that most directly impacts clients is latency. Writes are typically fast until Lustre's per-OST dirty page limit is reached. Reads are typically issued immediately and, when Lustre read ahead is triggered, will result in large RPCs. However, even for well-formed client RPCs, other factors can impact the per-request latency (e.g. server load from other jobs, on-disk and memory fragmentation on the server, and network problems).

Determining the latency distribution for high-level Lustre metrics and the key internal operations that drive them would benefit us in a couple of ways. First, looking at shifts in the distribution from the norm would allow us to determine when the system is performing poorly. Second, examining the latency of underlying operations will help drive root cause analysis. For example, slow writes can be caused by a number of different factors, including waiting for block allocations, slow disk controllers, and waiting for a journal checkpoint to complete.

It is not necessary for an application to do I/O in very large chunks because Lustre and the page cache will aggregate I/O. Typically, Lustre client nodes will do their best to aggregate I/O into 1 MB chunks and keep up to 8 I/O requests "in flight" at a time, per OST. There is a per *syscall* (Linux system calls) overhead for locking and such, so using 1 MB or larger read/write requests will minimize this overhead.

In view of the aforesaid, we have measured the latency for writing and reading 8 bytes of data on the Lustre file system. We used the Linux utility *Strace* to track all the system calls and signals for read and write operations under both NFS and Lustre [2-4, 17, 21]. In order to open a file to read or write 8 bytes and then close it, the following system calls are invoked by the Fortran run time library: *getcwd*, *open*, *ioctl*, *fstat*, *lseek*, *ftruncate*, *write,* and *close*. The *getcwd* function determines the path name of the existing directory. To open a file one uses the *fopen* function, which returns a file pointer. Once the file is opened, the file pointer is used to let the I/O library perform input and output operations on the file. *ioctl* is for device-specific read /write operations. The *fstat* function obtains information about an open file associated with the file descriptor and writes it to the area pointed to by the buffer. The *lseek* is to change the position of a file pointer. The *ftruncate* truncates the file. The *close* is to close the file.

To assess the overhead of write and read operations under Lustre, we also ran the *Strace* benchmark under NFS. We ran the benchmark five times and found that latency is almost constant. Figure 10 shows the average write latency for 8 bytes of data for each of these nine system calls on both Lustre and NFS file systems. Write latency for *open, fstat, ftruncate* and *write* on Lustre is higher by a factor of 1.6, 39.9, 3.6 and 2.0 than that on NFS respectively.

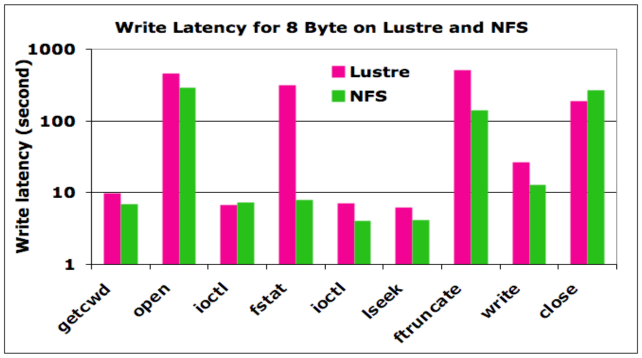

Figure 10. Write latency for 8 bytes on Lustre and NFS file systems.

Figure 11 show the average read latency for 8 bytes of data for each of these seven system calls on both Lustre and NFS file systems. Read latency for *fstat, read* and *close* on Lustre is higher by a factor of 83.1, 3.1 and 22 than on NFS respectively.

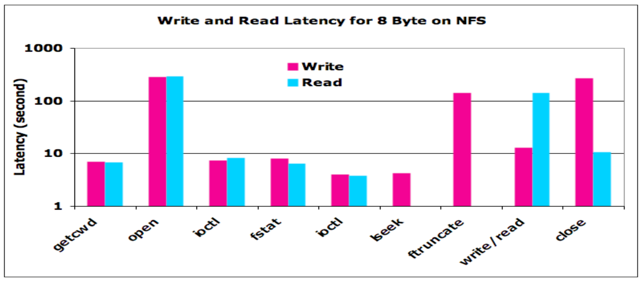

Figure 11. Read latency for 8 bytes on Lustre and NFS file systems.

C. *Modeling I/O Behavior of Applications*

In this subsection we model the I/O of four applications used in this paper (Overflow, Cart3D, USM3D and MITgcm). These four applications perform sequential I/O (reading grid file and writing checkpoint/restart file), i.e., all the processes send data (using *MPI-Send/Recv* or *MPI_Gather*) to rank 0, which writes it to the file. For read, rank 0 reads the data from file and then sends (using *MPI_Send/Recv* or *MPI_Bcast*) to other ranks. Memory of a node we studied is 24 GB. File size was chosen to be 8 GB and 56 GB to ensure that for 8GB data comes from memory cache and for 56 GB it comes from disk. We did not include application Enzo as it uses HDF5 format. In addition, we present results on multiple OSTs with various stripe sizes and block sizes to find an optimum set of Lustre parameters that can give the highest I/O performance for the NASA applications investigated in this paper.

1) *Performance on a Single OST*

We investigated the write and read policies on both NFS and Lustre on Pleiades. We ran the SWR benchmark for writing and reading 8 GB and 56 GB files. It may be recalled that the Pleiades Westmere node (12 cores) has 24 GB of memory. The kernel uses 1 GB and the rest is available for user applications. We wanted to investigate whether a write operation goes straight to disk or the data is stored in page cache before being written out and whether a read operation reads data from the disk or the memory buffer. Note that the I/O controllers have policies for both read and write operations:

Read Policy: The read policy dictates whether the controller reads sequential sectors of the logical drive when seeking data or not.

- Read-Ahead policy is one in which the controller reads sequential sectors of the logical drive prior to the issuance of the read instruction. This improves system performance if the data actually exists on sequential sectors of the logical drive.
- No-Read-Ahead policy is one where the controller does not use read-ahead policy.
- Adaptive Read-Ahead policy is one where the controller initiates read-ahead only if the two most recent read requests accessed sequential sectors of the storage disk. If subsequent read requests access random sectors of the disk, then the controller reverts back to no-read-ahead policy. The controller continues to monitor whether read requests are accessing sequential sectors of the disk or not, and can initiate read-ahead if necessary.

Write Policy: The write policy controls whether the controller sends a write-request completion signal as soon as the data is in the buffer cache or after it has been written to disk.

- Write-back caching is one in which the controller sends a write-request completion signal as soon as the data is in the controller cache but has not yet been written to disk. Write-back caching improves performance since subsequent read requests can more quickly retrieve data from the controller cache than they could from the disk. Write-back caching however entails a data integrity risk, since a system failure could prevent the data from being written to disk even though the controller has sent a write-request completion signal. In this case, data may be lost. Other applications may also experience problems when taking actions that assume the data is available on the disk.
- Write-through caching is one in which the controller sends a write-request completion signal only after the data is written to the disk. Write-through caching provides better data security than write-back caching, since the system assumes and reports that a write has been completed only after it has been safely written to the disk.

Figure 12 shows write and read bandwidth for 8 GB and 56 GB files. Write bandwidth is same for both 8 GB and 56 GB file: averages are 223 MB/s and 217 MB/s for 8 GB and 56 GB file respectively. On the other hand read bandwidth is 4853 MB/s and 370 MB/s for 8 GB and 56 GB file i.e. bandwidth for 8 GB file is higher than that for 56 GB file by a factor of 13. Clearly for 8 GB file, data is being read from page cache. On the other hand, for the 56 GB file, data is being read from disk as there is not enough memory on the node to cache the 56 GB file. The disparity between the write bandwidth for both file sizes and the read bandwidth for the 8 GB file (being read from cache) would seem to indicate a write-through caching policy is in effect. However, we know that this is not necessarily the case under Linux unless one is performing direct I/O. Some mechanisms in Lustre may be limiting the single process, single OST throughput, including the per-OST limit of 32 MB of dirty data and the per-OST maximum of eight outstanding RPCs at any given instance. Further testing is needed to find if one or a combination of these two factors, or possibly some other factor, is limiting write performance.

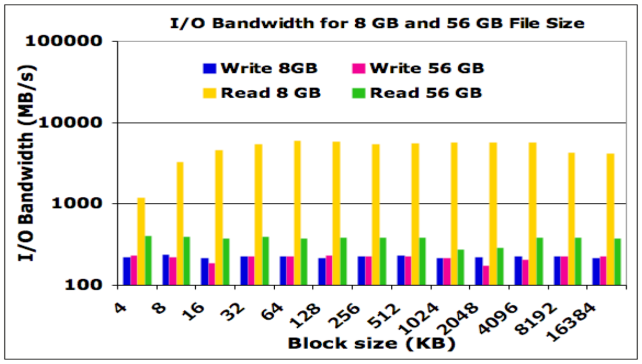

Figure 12. I/O bandwidth for 8 GB and 56 GB disk files.

Figure 13 shows the write bandwidth of a single writer to a single OST by varying the block size for writing 56 GB file on a disk. It is clear from this figure that write and read bandwidth does not depend on the block size (amount of data transferred per read or write call). The average write bandwidth on NFS and Lustre is 270 MB/s and 220 MB/s respectively – better by 19% on NFS than that on the Lustre. The reason for this is that the latency for *fstat* and *writes* is much higher on Lustre (see Figures 10). Read bandwidth on Lustre is better by 66% than on NFS (385 MB/s versus 232 MB/s respectively).

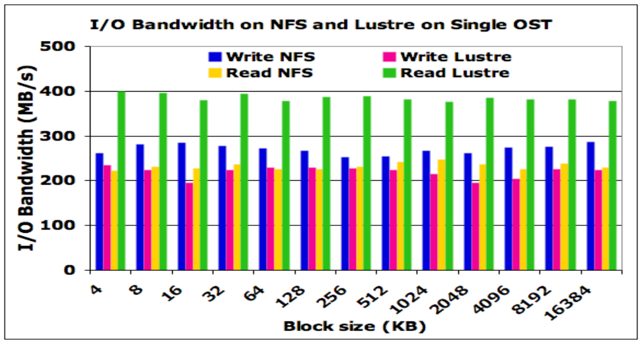

Figure 13. I/O bandwidth on NFS and Lustre using single OST.

2) *Performance on Multiple OSTs*

In this subsection, we present results for multiple OSTs with various stripe sizes and block sizes to find optimum Lustre parameters that can give the highest I/O performance. Figure 14 shows the write bandwidth on 16 OSTs and for various stripe sizes ranging from 1 MB to 64 MB and block sizes ranging from 4 KB to 16384 KB. Maximum write bandwidth is 714 MB/s for 1 MB block size and 32 MB stripe size. As mentioned earlier the write bandwidth for single OST is 220 MB/s so with 16 OSTs it has increased by a factor of 3.2. Lowest write bandwidth is for 4 KB block size and then it increases gradually until 256 KB. The reason for this is that there is more overhead for a small block size compared to a large block size. Figure 15 shows the corresponding results for read bandwidth. Maximum read bandwidth is 920 MB/s for 2 MB block size and 1 MB stripe size. It may be recalled that for single OST read bandwidth is 385 MB/s so with 16 OSTs it has increased by a factor of 2.4. Clearly, 16 OSTs increase both the write and read bandwidths, by factors of 3.2 and 2.4, respectively.

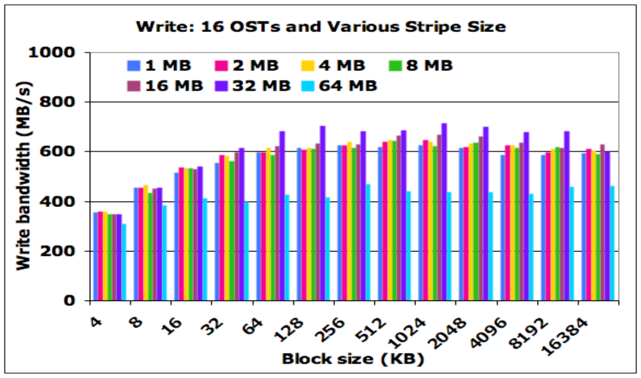

Figure 14. Write bandwidth for various stripe sizes on 16 OSTs.

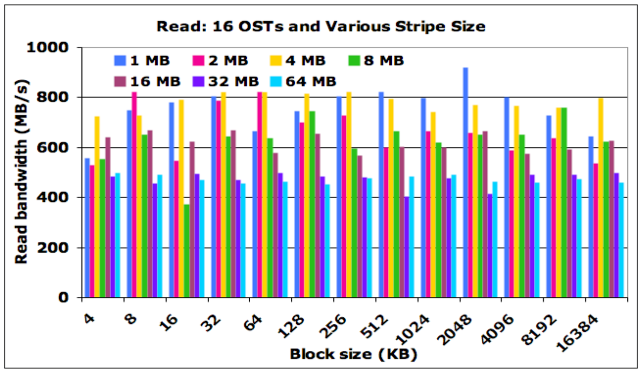

Figure 15. Read bandwidth for various stripe sizes on 16 OSTs.

Figure 16 shows the I/O (write and read) bandwidth for single OST and 16 OSTs. We notice that write bandwidth is much better with 16 OSTs compared to that on a single OST. Figure 17 shows the percentage improvement of I/O on 16 OSTs relative to 1 OST. Maximum percentage improvement is 239% and 170% for write and read respectively. Maximum for write and read is for a block size of 1024 KB and 256 KB respectively.

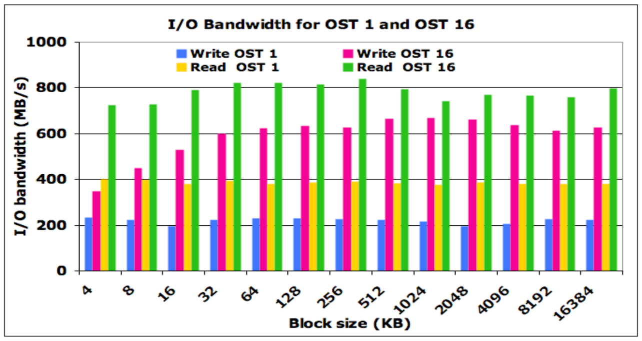

Figure 16. I/O bandwidth single OST and 16 OSTs.

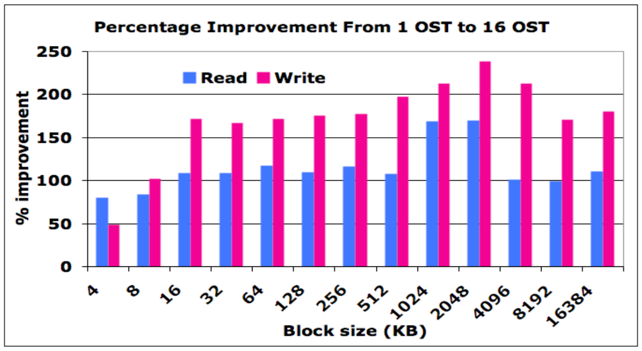

Figure 17. Percentage I/O improvement with 16 OSTs over 1 OST.

D. *File per Process vs. Single Shared File*

In this section we compare the performance of file-per-process and single-shared-file approaches using IOR in POSIX mode. We used a transfer size of 16 MB and block sizes ranging from 16 MB to 8 GB. We used 8 cores per node). The aggregate file size ranged from 128 MB to 128 GB. All the tests were conducted in non-dedicated mode, i.e. with users running on other parts of the system. To account for this, we ran each test 5 times and used the maximum performance rate (MB/s).

In measuring the I/O performance, file caching due to page cache results in measured read bandwidth rate to be very high as the data is being buffered in memory rather than being written directly to the disk. The page cache uses unused memory of a compute node to buffer I/O transactions and flush them to disk later on to improve I/O performance for small files. In order to avoid the caching effect on I/O performance, we used the I/O files to be much larger than compute node memory size. As noted before, each Pleiades Westmere compute node has 24 GB of memory.

Figure 18 shows the measured aggregate I/O bandwidth for 8 processes using one file per process and single shared file strategy for different aggregate file sizes. Aggregate file sizes was changed by changing the block size, i.e. aggregate file size = *BlockSize**NumTasks*, while *TransferFile* size was fixed at 16 MB. When the file size is small (4 GB or less), file caching has a considerable effect on the performance. For read bandwidth there are clearly two performance regions on Pleiades. When the file size is from 128 MB to 4 GB, read bandwidth for both single file per process and single shared file ranges from 5 GB/s to 9 GB/s, which shows that the data clearly remains in the page cache after it is written. During this regime, the read performance corresponds to the memory read performance. As the file size increases, the page cache can no longer hold all the data and the read operation must get the data from the disk. The read performance degrades and gradually becomes stable when all data access is from disks. When the read data comes from disk, read bandwidth is 1 GB/s and 250 GB/s for file-per-process and single shared file respectively. We see no memory buffer or caching effect on write bandwidth for both single-file-per-process and single shared file and write rates vary from 790 MB/s to 1.295 GB/s and 251 MB/s to 204 MB/s respectively. Write bandwidth for single file per process is higher than single shared file by a factor of 3.1 to 6.3.

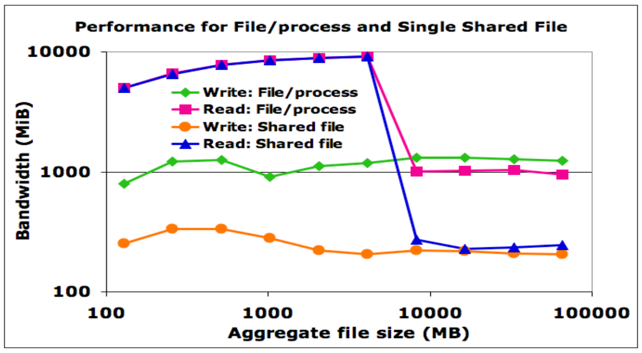

Figure 18. I/O bandwidth of file per process and shared file.

IX. CONCLUSIONS

In this work, we analyzed the NASA scientific and engineering workload to develop a better understanding of the I/O strategies used by a diverse array of applications. We developed a performance metric tool to determine the RPC size distribution; useful information to pinpoint the bottlenecks because more I/O done using small RPCs points to inefficient use of Lustre file system. RPC size distribution measures the efficiency of the Lustre system. Knowledge of number of file opens and closes enables an application scientist to use I/O strategies to increase the performance of I/O in their applications. Information about amount of I/O data helps users choose the optimal stripe size and stripe counts to enhance I/O performance. The extracted statistics are useful in determining the I/O pattern of the application and can assist in identifying possible improvements of users applications.

We also measured the overhead associated with write and read and write operations on both NFS and Lustre system. We examined the read and write policies on Pleiades Lustre file system and found that when compared to read performance, write performance behaved as if it were being performed using a write-through caching policy, and reads can have a higher performance when they come from page cache. Finally, we also investigated the "cache effect" on the I/O operations.

We characterized the Pleiades Lustre file system to determine the optimal stripe size and stripe counts, which enhance the performance of the applications significantly. We studied the I/O performance for single file per rank and single shared file accessed by all the ranks on Pleiades. We have shown that the I/O performance on Pleiades is highly dependent on file access type, access pattern, size, and I/O transaction size. We found that performance of file per process is much better than that using single shared file.

It is clear that increasingly larger-scale supercomputers will require that application developers examine the I/O capabilities that will be available to them and determine how best to utilize them.

References

[1] S. Lang, P. Carns, R. Latham, R. Ross, K. Harms, and W. Allcock, "I/O performance challenges at leadership scale," in Proceedings of Supercomputing, SC09, November 2009.

[2] Lustre: http://wiki.lustre.org/index.php/Main_Page

[3] Lustre Operations Manual –Version 1.8, http://wiki.lustre.org/index.php/Lustre_Documentation

[4] F. Wang, S. Oral, G. Shipman, O. Drokin, T. Wang, and I. Huang. Understanding Lustre filesystem internals. Technical Report, ORNL/TM-2009/117, Oak Ridge National Lab., National Center for Computational Sciences, 2009. http://wiki.lustre.org/index.php/Lustre_Center_of_Excellence_at_Oak_Ridge_National_Laboratory

[5] OpenSFS: Open Scalable File system Inc., http://www.opensfs.org/

[6] S. Saini, D. Talcott, R. Thakur, P. A. Adamidis, R. Rabenseifner, and R. Ciotti, "Parallel I/O Performance Characterization of Columbia and NEC SX-8 Superclusters," in IPDPS, 2007.

[7] J. Borrill, L. Oliker, J. Shalf, and H. Shan," Investigation of leading HPC I/O performance using a scientific-application derived benchmark," in SC, 2007.

[8] J. Borrill, L. Oliker, J. Shalf, H. Shan, and A. Uselton," HPC Global File System Performance Analysis Using a Scientific-Application Derived Benchmark," Parallel Computing, vol.35, no.6, pp. 358–373, 2009.

[9] W. Yu, J. Vetter, and S. Oral. Performance characterization and optimization of parallel I/O on the Cray XT. In *Proceedings of 22nd IEEE International Parallel and Distributed Processing Symposium (IPDPS'08)*, Miami, FL, 2008.

[10] Performance Co-Pilot User's and Administrator's Guide and Performance Co-Pilot Programmer's Guide, http://oss.sgi.com/projects/pcp/documentation.html

[11] PMDA: Performance Metrics Domain Agent, http://techpubs.sgi.com/library/manuals/4000/007-4993-004/sgi_html/ch05.html

[12] Overflow, http://aaac.larc.nasa.gov/~buning/

[13] D. J. Mavriplis, M. J. Aftosmis, and M. Berger. High Resolution Aerospace Applications using the NASA Columbia Supercomputer, Proc. ACM/IEEE SC05, Seattle, Washington, Nov. 2005.

[14] USM3D: http://tetruss.larc.nasa.gov/usm3d/

[15] M.I.T General Circulation Model (MITgcm), http://mitgcm.org/

[16] Enzo Version 2.0, http://enzo.googlecode.com

[17] Linux NFS Overview: http://nfs.sourceforge.net/

[18] Pleiades. http://www.nas.nasa.gov/hecc/resources/pleiades.html

[19] IOR HPC Benchmark: , http://sourceforge.net/projects/ior-sio

[20] PMVAL: Performance Metrics Value Dumper, http://oss.sgi.com/projects/pcp/man/man1/pmval.1.html

[21] *strace*(1): trace system calls/signals - Linux man page http://linux.die.net/man/1/**strace**, http://sourceforge.net/projects/strace/