Performance Characteristics of the Lustre File System on the Cray XT5 with Respect to Application I/O Patterns.

Lonnie D. Crosby, National Institute for Computational Sciences

ABSTRACT: As the size and complexity of supercomputing platforms increases, additional attention must be given to the performance challenges presented by application I/O. This paper presents the performance characteristics of the Lustre file system utilized on the Cray XT5 system and illuminates the challenges presented by applications which utilize tens of thousands of parallel processes.

KEYWORDS: Cray XT5, IOR, Lustre, IO

1 Introduction

The increasing size and capability of modern supercomputing platforms points to promises of ever increasing scientific production. The major metric in categorizing these systems and their scientific potential is their peak performance measured in floating-point operations per second (FLOP/s). For example, two Cray XT5 systems housed at Oak Ridge National Laboratory designated Jaguar and Kraken measure in the high teraflop (TF) to petaflop (PF) range. This metric, however, is only a function of the performance and number of processors within the system and should only be viewed as a theoretical maximum.

Application performance depends on many more factors other than processor performance and number. For example, an application will need to interact with various aspects of the supercomputing platform in order to produce scientific results. Besides performing raw operations, an application will need to access both local and remote random-access memory (RAM), communicate with local and remote processes, and perform I/O to acquire raw data and report results. Thus, interactions with RAM, network interconnect, and file systems must be addressed to understand application performance.

The focus of this paper is the application's interaction with the file system. This interaction is truly a cooperative exercise. Both parties, application and file system, are equally responsible for ensuring adequate performance. By understanding the application's I/O patterns and the features/limitations of the file system, optimal performance can be realized.

2 Application I/O

Applications perform I/O in a variety of ways. Many of the particular choices depend heavily on the nature of the specific application. Although this topic is very application specific, general I/O patterns that accurately describe portions of real application I/O patterns can be identified. The fundamental questions regarding I/O patterns are:

- How many processes are performing I/O?
- How is this I/O performed?

The first question relates to the number of process which are involved in I/O operations. In extreme circumstances, an application may only utilize one process in I/O or may involve all processes. The second question gives some additional information. An application may perform I/O by writing a single file in chunks of a given size. This file may be uniquely owned by a single process or shared between many processes. It is important how these processes interact in I/O operations. For shared files, this interaction may be collective or independent.

As a result, three distinct application I/O patterns will be investigated. The first refers to an application which only uses one process to perform I/O. The second refers to an application in which all processes perform I/O by writing an individual file per process. The third I/O pattern also refers to an application in which all processes perform I/O. However, these processes will share access to a single file. It should be noted that none of these I/O patterns should be considered as optimal. An optimal I/O pattern for an application is very dependent on the specifics of the application and is most likely some combination of the above patterns.

3 Lustre File System

The Lustre file system [2] is a parallel shared file system which is available on the Cray XT5 supercomputer at the National Institute of Computational Sciences (NICS). This file system make use of dedicated I/O servers called Object Storage Servers (OSS) which each manage a set of drives which are called Object Storage Targets (OST). A metadata server (MDS) manages every file on the file system by keeping track of which objects are associated with each file. Objects are managed by the OSSs and stored on the OSTs. A file may be striped across multiple OSTs by assigning it multiple objects which reside on different OSTs. Sequential OSTs are managed by different OSSs in order to maintain a balanced workload. The file is striped between objects by assigning a stripe size. As a portion of a file is written to an object, if its size becomes larger than the stripe size the next object is utilized.

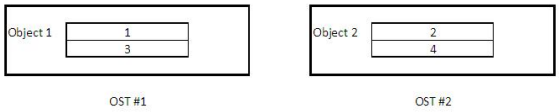

Figure 1: A diagrammatic representation of a file striping pattern utilizing two OSTs, a 1 MB stripe size, and a 4 MB file.

This is shown in Figure 1 for a 4 MB file which is associated with two objects with a stripe size of 1MB. The Lustre file system can be controlled by setting the number of objects associated with the file (stripe count) and the size of the stripes (stripe size). Additionally, the beginning OST index (stripe index) may be set to select the starting OST from which objects will be assigned. Additional objects will be assigned to sequential OSTs. These parameters may be set for both files and directories on the file system. Files will inherit the stripe settings of the directory in which they reside or may be given individual stripe attributes. These parameters are set by the "lfs setstripe" command. They may be applied to directories at any time; however, the attributes do not apply to files which are already present in the directories. The stripe attributes of files are set at file creation and must be assigned prior to this action.

4 Hardware

The goal of this work is to quantify the performance characteristics of the previously stated three application I/O patterns on the Lustre file system with emphasis on their interaction. These tests of applicaiton/file system performance are performed on the NICS supercomputer "Kraken" which is a Cray XT5 platform. This supercomputer consists of 8253 compute nodes which contain two quad-core 2.3 GHz AMD Opteron processors. The peak performance of this system is about 608 TF. A mixture of available RAM is available with both 8 and 16 GB nodes utilizing 800 MHz DIMMS. The internodal communication network consists of a 22x16x24 3D Torus topology which utilizes the SeaStar 2+ interconnect. This high speed network is capable of achieving a maximum bi-directional bandwidth of 9.6 GB/s.

The Lustre file system consists of 1 MDS, 48 OSSs, and 336 OSTs. The OSSs are distributed evenly along two dimensions of the torus topology. The peak bandwidth of the filesystem is about 30 GB/s, which is uniquely determined by the number of OSSs. Each OSS manages 7 OSTs which have a usable size of 7.2 TB through 10 1 TB disks in a (8+2) RAID 6 configuration. The Lustre software utilized is based on version 1.6.5.

The initial observation from this combination of compute nodes and file system is the proportion of computational resources to file system resources. Given the 30 GB/s peak bandwidth of the Lustre file system, if every compute core is equally splitting the file I/O bandwidth each core would maintain less than 500 KB/s. Similarly each compute node would then only be able to maintain about 4 MB/s. Clearly, a cooperative balance between the number of cores/nodes performing I/O and the parallel capabilities of the file system are necessary for acceptable I/O performance.

5 I/O Tests

These tests are performed with IOR version 2.10.1 [1] from Lawrence Livermore National Laboratory. Unless otherwise noted the POSIX API is utilized in these tests. Other APIs utilized include MPI-IO version 2.0 [4] and parallel HDF5 version 1.6.2 [3]. IOR gives the flexibility to benchmark I/O by altering the size of read and write operations (transfer size), the number of processes, the structure of parallel files (block size, repetitions, and file offsets), and the mode of operation (file per process and single shared file). All tests utilize direct I/O in which the system level buffers are bypassed which gives finer control over the characteristics of the I/O operations. These tests are performed utilizing three separate trials of both read and write operations in which the maximum read and write performance is reported. This method yields data which gives the expected best case conditions given some variability due to the file system load. These tests are run on a non-dedicated system. Therefore, measured bandwidths are generally substantially less than peak due to external file system load.

5.1 Single Writer

An application which utilizes one process that performs I/O to a single file must, in general, aggregate data from all other processes. This situation imposes some overhead from the associated communication and memory costs. Apart from these considerations to application performance, a single process establishes a single I/O stream to the filesystem. It is the interaction of this process and the Lustre file system which is of interest in this case. The comparable test is to determine the effectiveness of file striping on the resulting single file.

The striping pattern of this file is scanned between stripe counts of 1 and 160, which is the maximum allowed within Lustre. A file size ranging between 32 MB and 5 GB is utilized to keep the load per OST constant at 32 MB. This method is utilized in order to eliminate increases in performance due to a decreased load per OST, which would be the case with a constant file size. A 32 MB transfer size is utilized in these tests in order to give increased I/O volume. A larger transfer size is consistent with the I/O aggregation necessarily, although, given the particular application may vary in size. The stripe size is also changed between 1 and 32 MB to determine the effect of this parameter on performance. Figures 2 and 3 show the write and read performance results.

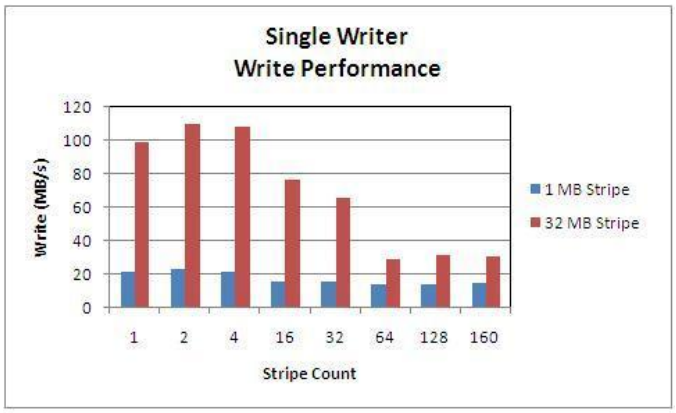

Figure 2: The write performance measured in MB/s for a single process as a function of stripe count.

Figure 2 shows that striping a single file which is written by a single process over multiple OSTs doesn't provide substantially improved performance. The maximum write performance is obtained with 2 OSTs. However, this performance gain is not significant compared to utilizing 1 or even 4 OSTs. The largest gains in performance come from increasing the stripe size to 32 MB. This operation provides for a more contiguous write on the particular OST as compared to a 1 MB stripe which may be distributed throughout the device. A similar situation is seen in Figure 3 with respect to read performance. The only difference, besides the increased bandwidth as compared to writes, is that with a 32 MB stripe size stripe counts up to 16 provide increased performance. These results suggest that performance is limited by the single process which performs I/O.

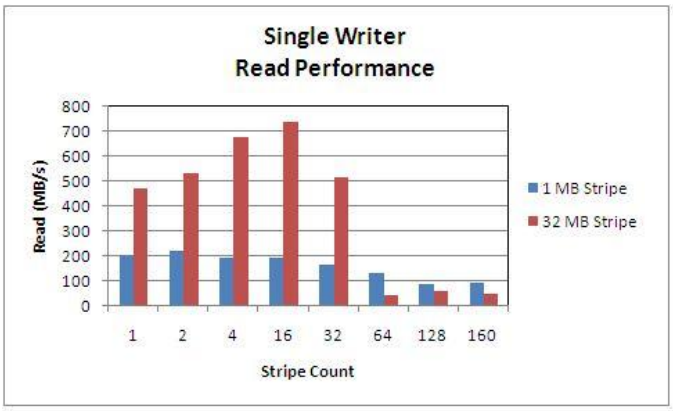

Figure 3: The read performance measured in MB/s for a single process as a function of stripe count.

Due to the increased performance of a 32 MB stripe, which happens to be equal to the transfer size, another test is necessary to correlate the transfer size and stripe size utilized. This test will restrict a 128 MB file to a single OST. The transfer size is varied between 1 and 32 MB while the stripe size is varied between 1 and 128 MB. The write performance of these tests are shown in Figure 4. For a 1 MB transfer size performance is unaffected by changes in the stripe size. The performance is effectively limited by the 1 MB transfer size. However, the 8 and 32 MB transfer size cases have performance which increases with the stripe size. Previous studies on the Cray XT4 [5] have shown a very strong interdependence between transfer size and performance. In general at stripe sizes smaller than the transfer size performance is limited by the stripe size. At larger stripe sizes performance is limited by the transfer size. The maximum performance for the 8 MB transfer size case occurs at a stripe size of 32 MB. Similarly, the maximum performance for the 32 MB transfer size occurs at a stripe size of 64 MB. Although stripe sizes slightly larger than the transfer size can provide increased performance, a stripe size roughly equal to the transfer size should provide a good compromise which would reduce the possibility that either will become a substantial bottleneck to performance.

Figure 4: The write performance for a single process as a function of transfer and stripe size.

5.2 Single Shared File

The single shared file application I/O pattern utilizes multiple processes performing parallel I/O to a single shared file. This results in multiple I/O streams which access the same file. The structure of this parallel file is setup with offsets for each process such that they do not access the same portion of the file. Important application I/O characteristics include the transfer size and layout of the parallel file. From previous experience and results from the single writer performance, this shared file should be striped over a number of OSTs which are at least equal to the number of processes involved in the I/O operation. This choice will minimize both overhead associated with splitting an operation between OSTs and the contention between processes over a single OST.

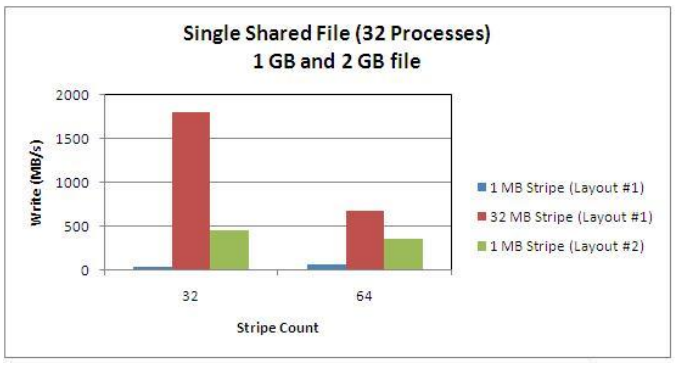

Figure 5: The write performance of a single shared file.

Figure 5 shows the results of this test utilizing 32 processes which write to a 1 or 2 GB shared file. This file is striped over 32 and 64 OSTs. The total file size is altered by increasing the amount of IO each process performs (block size) to maintain 32 MB per OST. This is an attempt to keep the load on each OST constant. A 32 MB transfer size is also utilized in this test. The layout of the shared files are shown in Figure 6 [5]. Sequential offsets are utilized in both layouts. Layout #1 uses a stride of 32 or 64 MB and layout #2 uses a stride of 1 or 2 MB with 32 repetitions of the overall pattern in order to maintain the same total file size and size per process. For layout #2 a 1 MB transfer size is utilized.

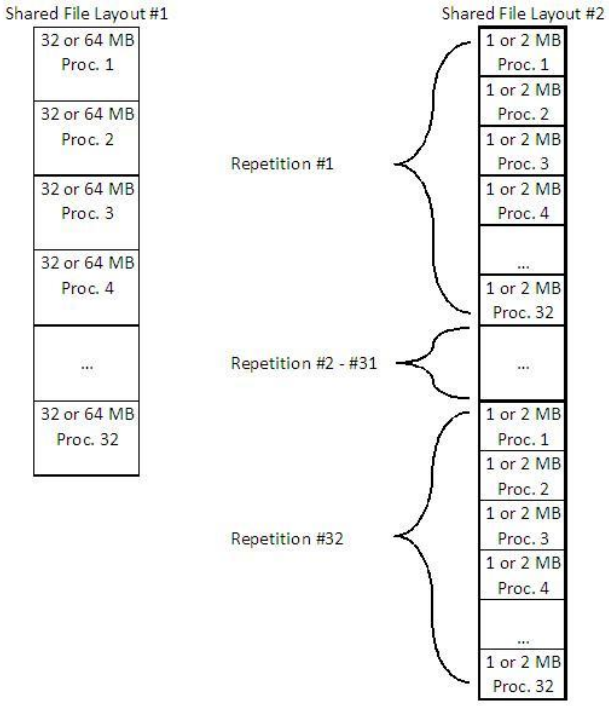

Figure 6: A diagrammatic representation of two distinct parallel file layouts.

Diminished performance is seen when utilizing a 1 MB stripe size with shared file layout #1. There are two reasons for this occurrence. The performance is limited by the 1 MB stripe size and portions of the file written by each process are stored on each OST. Each process is writing to every utilized OST which causes process contention for any particular OST and incurs additional overhead related to switching OSTs. Both of these situations can be overcome by utilizing a 32 MB stripe size. This localizes each processes I/O to a single OST which minimizes process contention and overhead. Using a stripe count of 32 provides better performance than a stripe count of 64 due to the overhead of switching between OSTs for each process. Process contention for an OST is not a factor in this reduction of performance.

The vastly improved performance from utilizing a 32 MB stripe size comes from both data localization and

data continuity as compared to the 1 MB stripe size case. To gauge the relative importance of these two factors shared file layout #2 is used with a 1 MB stripe size. This combination provides for data localization but not data continuity on an OST. As Figure 5 shows this results in improved performance, but only accounts for 25% to 50% of the overall increase in performance seen by using a 32 MB stripe size. For shared files, both data locality and continuity are of importance to I/O performance.

5.3 File per Process

From the information obtained during the tests on the single writer application I/O pattern, it was determined that the resulting files should reside on an individual OST. Also the stripe size should be set in a manner consistent with the transfer size. The performance will be reduced if either of these two parameters become limiting. Setting these to be about equal should give near optimal performance for this case. The file per process application I/O pattern scales the idea of a single writer to encompass all processes. Each process will perform I/O operations on separate files. Figure 7 shows the write performance of this I/O pattern as a function of the number of processes/files. To be consistent with previous results each file is striped over a single OST which is chosen on a round-robin or load-based metric. Each file is 128 MB in size and utilizes a 32 MB transfer size. Both 1 MB and 32 MB stripe sizes are utilized.

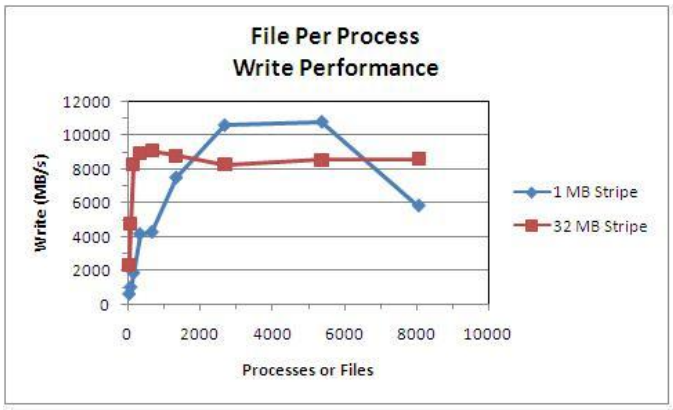

Figure 7: The write performance of the File per Process I/O pattern as a function of the number of processes/files.

In both cases the overall trend shows increasing performance as the number of processes/files are increased until a maximum is reached. This trend has been previously reported for the Cray XT3/XT4 platforms [5, 6] At large process counts, the performance degrades substantially for the 1 MB stripe size. For the 32 MB stripe size case the performance mostly flattens out at large process counts. In general, the 32 MB stripe size case performs better than the 1 MB case which is consistent with previous results.

There are two major implication to scaling this I/O pattern to large core counts. The first is the number of OSTs, which is 336 on Kraken. Increasing the number of processes/files is sustainable for some time after all the OSTs are utilized. This continues until contention for the file system resources become a limiting factor. The second is the number of simultaneous I/O operations, which are independent of the number of OSTs. The major characteristic of this situation is elongated times for file open operations due to contention at the MDS server. Additional contention may also be present at the OSSs and OSTs.

5.4 Scaling of the Single Shared File

From previous investigation of the single shared file application I/O pattern, it was determined that data locality and continuity are the most important factors affecting performance. As this pattern is scaled to larger process counts a trend similar to the file per process pattern emerges.

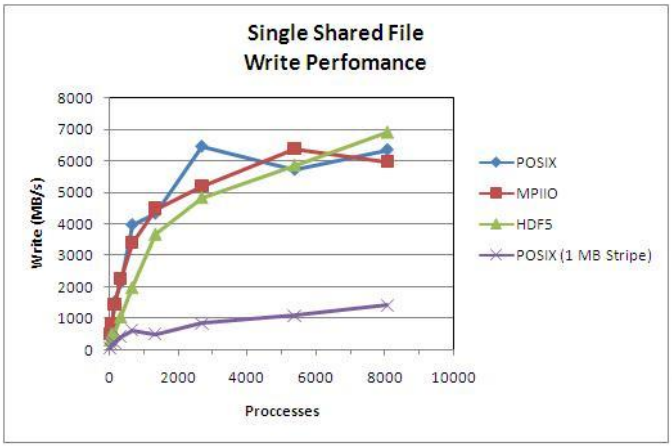

Figure 8: The write performance of the single shared file I/O pattern as a function of the number of participating processes.

This is shown in Figure 8 for the POSIX, MPI-IO, and HDF5 APIs. These tests utilize a file size of between 1.25 and 252 GB and the parallel file labeled as layout #1 in Figure 6 with a 32 MB block size. The parallel files are striped across a number of OSTs which are equal to the number of processes. Since the maximum stripe

count for the Lustre file system is 160, the stripe count is set to 160 for clients greater than 160. For data locality and continuity reasons the transfer size is set to 32 MB.

The performance of the three parallel I/O libraries are very similar and peak near 8k processors at 5983 to 6886 MB/s write bandwidth. The predominate feature is the leveling off of write performance at large process counts. The major limiting factor is the restriction to 160 OSTs. Performance increases dramatically at low process counts and continues to grow past 160 processes until the contention for these OSTs becomes limiting. These results are very similar to previous studies performed on the Cray XT4 [5]

The importance of data locality and continuity is also seen if Figure 8. By changing the stripe size to 1 MB data from each process is spread across OSTs. This has a very large negative impact on write performance on the order of 4000 MB/s, although the overall trend is similar to the 32 MB stripe size case.

6 Conclusion

In these tests of I/O performance, the major considerations are data locality and continuity. Previously proposed I/O guidelines [6] from benchmarks run on Cray XT3/XT4 systems agrees with the implications of focusing on these considerations. Data locality is achieved by restricting a process's I/O to a single OST for both private and shared files. For shared files, this locality must be ensured by a combination of parallel file layout and stripe characteristics. Data continuity is achieved by addressing the issue at both the application and file system level. The size of the I/O transfer is very important in terms of performance and may be adjusted in the application or by the use of external buffers. Larger I/O transfers perform substantially better than smaller transfers. If these large transfers are written to a single object in the Lustre file systems then data continuity can be maintained. However, the striping characteristics can easily deteriorate data continuity by spreading this data between objects or distributing it throughout a single OST. For this reason the I/O transfer size must cooperate with the file's striping characteristics such that data continuity is maintained. In general, stripe sizes less than application transfer sizes will degrade performance as mentioned previously. Stripe sizes larger than transfer sizes may also show improved performance; however, the overhead involved in completing a partial stripe eventually degrades performance.

Both the file per process and single shared file I/O patterns have distinct problems when scaled to high process counts. These methods are limited by either the physical number of OSTs or by the maximum Lustre stripe count of 160. Typically process counts of about 6k are sustainable for a file per processes pattern. This limit decreases somewhat for a single shared file to about 4k processes. Improvements in performance at high process counts may be realized if slightly different I/O patterns are utilized. A subsetting approach may be utilized which aggregates I/O from a subset of processes to a single process. For an application of many thousands of processes there may be many such I/O aggregators. In fact the MPI-IO API is able to perform this kind of aggregation through the use of some environmental variables.

A similar I/O pattern with respect to shared files is also possible. The application may divide itself into groups of processors which each write to an individual shared file. No additional memory costs are incurred during this process. The major benefit of this approach is the ability to utilize more of the filesystem by striping multiple shared files over disjoint sets of OSTs, which would circumvent the 160 OST restriction for a single file.

7 About the Author

Lonnie D. Crosby is a computational scientist in the National Institute for Computational Sciences (NICS) located at Oak Ridge National Laboratory (ORNL). NICS is a joint partnership between the University of Tennessee and ORNL. Lonnie has a Ph.D. in chemistry from The University of Memphis located in Memphis, TN. He may be contacted at Oak Ridge National Laboratory, P.O. Box 2008 MS6173, Oak Ridge, TN 37831-6173, Email: lcrosby1@utk.edu.

References

[1] IOR 2.10.1, https://asc.llnl.gov/computing_resources/purple/archive/benchmarks/ior/

[2] Lustre File System: High-Performance Storage Architecture and Scalable Cluster File System, White Paper, October 2008, www.sun.com/software/products/lustre/docs/lustrefilesystem_wp.pdf

[3] HDF5 1.6.2, http://www.hdfgroup.org/HDF5

[4] MPI-2: Extensions to the Message Passing Interface. http://www.mpi-forum.org/docs/mpi-20-html/mpi2-report.html

[5] Using IOR to Analyze the I/O performance for HPC Platforms, H. Shan and J. Shalf, CUG Proceedings 2007.

[6] Guidlines for Efficient Parallel I/O on the Cray XT3/XT4, J. Larkin and M. Fahey, CUG Proceedings 2007.