Using IOR to Analyze the I/O performance for HPC Platforms

Hongzhang Shan[1], John Shalf[2]

National Energy Research Scientific Computing Center (NERSC)[2]
Future Technology Group, Computational Research Division[1]
Lawrence Berkeley National Laboratory
{ hshan@lbl.gov , jshalf@lbl.gov }

Abstract

The HPC community is preparing to deploy petaflop-scale computing platforms that may include hundreds of thousands to millions of computational cores over the next 3 years. Such explosive growth in concurrency creates daunting challenges for the design and implementation of the I/O system. In this work, we first analyzed the I/O practices and requirements of current HPC applications and used them as criteria to select a subset of microbenchmarks that reflect the workload requirements. Our analysis led to selection of IOR, an I/O benchmark developed by LLNL for the ASCI Purple procurement, as our tool to study the I/O performance on two HPC platforms. We selected parameterizations for IOR that match the requirements of key I/O intensive applications to assess its fidelity in reproducing their performance characteristics.

1. Introduction

The advent of petascale computing is leading to HEC platforms of unprecedented concurrencies. Within the next three years, platforms will be built with unprecedented concurrencies that may, in some cases include over a million computational cores. This daunting level of concurrency will pose enormous challenges for future I/O systems that must support efficient and scalable the data movement between disks and distributed memories. In order to guide the design of the new underlying I/O system, we need to gain a better understanding of applications requirements. However, it is impractical to run the full-fledged applications for testing and evaluation of new I/O solutions. Therefore, we also need to select a compact proxy benchmark that is capable of emulating both the disk access patterns of a diverse workload.

In this paper, we describe the results of a comprehensive analysis of the I/O requirements and usage patterns at the National Energy Research Supercomputing Center (NERSC). We describe how the workload analysis fed into the selection of the LLNL IOR benchmark to emulate elements of the NERSC workload. We describe a variety of pitfalls for developing sensible and reproducible IO benchmark results. Finally, we describe our analysis of the disk access patterns of a selection of I/O intensive applications, and how to select suitable parameters for IOR to emulate their behavior. We demonstrate that with suitable parameterizations, IOR is capable of closely approximating the performance and behavior of the original application.

2. Workload Analysis

We conducted a workload assessment that studied the current practice and future requirements of I/O systems for the NERSC user community. Based on the project descriptions in yearly allocation requests (ERCAP), 50 I/O intensive projects were selected from field over 300 allocation requests for the NERSC computing platforms. For this subset of 50 projects, each PI was asked to fill a detailed questionnaire regarding their current I/O practices and future requirements of their applications. We also performed some application drilldowns and performance studies to provide a more detailed picture of application I/O requirements. The major results of this study include:

- Random access is rare; the I/O access is dominated by sequential read/write.
- Application I/O is dominated by append-only writes.
- I/O transactions sizes vary widely: from several Kilobytes to tens of megabytes.
- The majority of applications have adopted a one-file-per-processor approach to disk-IO (POSIX I/O for Fortran unformatted I/O) rather than using parallel I/O APIs (such as MPI-IO).
- Most applications use their own custom file-formats rather than portable self-describing formats such as NetCDF or HDF5, but interest in these formats is growing.

From a system level, the I/O activity is nearly equal between reads and writes – with some dominance by the reads. However, the system-level activity includes data movement to and from the archival storage systems. With a narrower focus on parallel applications, the data flow for writes is more dominant. The following factors also contribute to the dominance of writes in application I/O activity: 1) In most cases, users will transfer the result files to other machines for post-processing or visualization analysis and not on the same platforms on which the computation has been done. 2) Users frequently output data to files for checkpointing or restart purpose. Most of these files may never need to read back. 3) Input files to initialize the applications are often small – particularly when the input conditions are automatically generated by the code from the parameters supplied in the input-deck.

The majority of users continue to embrace the approach that each process uses its own file to store the results of its local computations. An immediate disadvantage of this approach is that after program failure or interruption, a restart must use the same number of processes. A more serious problem is that this approach does not scale, and leads to a data management nightmare. Tens or hundreds of thousands of files will be generated on petascale platforms. A practical example [15] is that a recent run on BG/L using 32K nodes for a FLASH code generated over 74 million files. Managing and maintaining these files itself will become a grand challenging problem regardless of the performance. Using a single or fewer shared files to reduce the total number of files is preferred on large-scale parallel systems.

Most users still use the traditional POSIX for Fortran77 unformatted IO (usually implemented on top of POSIX) interfaces to implement the I/O operations. The

traditional POSIX/F77 serial interfaces are not designed for the large-scale distributed memory systems. Each read/write operation is associated with only one memory buffer and cannot read/write a distributed array together. If the application has complex data structures, this simple interface may cause significant inconvenience for application users to reassemble the data files for the purpose of data analysis and visualization. In addition, the parallel file reassembly process tends to be implemented serially, which performs very poorly on cluster filesystems.

Even worse, we found that some users assign one process to handle all I/O operations. The process, typically with MPI rank = 0, is responsible for collecting the data from all other processes and writing it incrementally to the file, or for distributing the data to other processes after it has read the data from the file. This practice not only limits the data size to access (due to memory size limitation accessible to the responsible process) but also serializes the I/O operations and significantly impacts the I/O performance.

Concurrent access to a single file using parallel I/O APIs such as MPI-IO [9] is slowly emerging. This trend is motivated by using fewer files and will greatly simplify the data analysis and archival storage. Ultimately, the users would like to have the same logical data organization within the data file, regardless of how many processors were involved in writing the file. However, there continues to be considerable resistance to this approach due the perception that parallel I/O is less efficient than one-file-per-processor. Some of the perception is derived from user experiences on non-parallel IO systems such as NFS, but we hope to dispel many of those rumors with data collected on modern parallel filesystems such as GPFS and Lustre.

Some users are beginning to adopt advanced file formats, such as HDF5 [10] and parallel NetCDF [7,11] to increase the portability, enable file format evolution without breaking older file readers, and improve data provenance. However, users also have a perception that these higher-level file formats are more difficult to program and will cause significant performance loss compared with the traditional POSIX interface. In this report, we quantify the amount of overhead incurred by using these higher-level file formats.

The data size of each I/O operations varies widely from small to very large (several KB to tens of MB) on a per-application basis, which argues for a benchmark that is highly parameterized to cover the range of application behaviors. However, small transactions and random accesses result in very poor I/O performance and are often implicated as the primary performance bottleneck in poorly written I/O implementations. Therefore we worry that many of the applications that exhibit very small transaction sizes and gather/scatter I/O behavior are not employing best-practices in the design of their I/O. Designing an I/O subsystem around the raw statistical description of the I/O patterns of the NERSC workload without understanding the intentions of the application developers may result in selecting filesystem based on the requirements of poorly written implementations! Therefore, we are slowly working through a deeper analysis of key I/O intensive applications in order to select an even smaller subset that conform to best practices in order to motivate future I/O system requirements.

Our selection of a proxy IO benchmark figures heavily into this analysis process because it is used to set our expectations for the application I/O performance so that we can validate our understanding of its behavior. Changing the I/O implementation to favor larger transactions requires examination of the intended (logical) data layout, to see whether the application is misusing the filesystem, or if the application requirements demand use of small/scattered disk accesses. In most cases, we have found that poorly formulated I/O patterns are indicative of a misunderstanding of the inefficient data patterns that would presented to the filesystem, and that simple changes to the implementation could result in dramatic improvements in performance.

3. Benchmark Selection

Our goal is to select (or write) a benchmark that is capable of emulating the full range of workload characteristics that we identified in the survey of NERSC applications. Based on the survey results, we characterized the I/O requirements of these applications into following parameters: access pattern, file type, I/O transaction size, file size, concurrency, and programming interface. We examined a wide variety of publicly available and actively maintained I/O benchmarks [1,2,3,4,5,6,12]. We found that most of the existing benchmarks are not reflective of the current I/O practice of HPC applications, either because the access pattern did not correspond to that of the HPC applications, because they only exercise POSIX APIs (eg. no MPI-IO, HDF5, NetCDF), or because they measure only serial performance.

Ultimately our benchmark survey determined that LLNL's IOR benchmark met all of our requirements for a parameterized benchmark that reflects HPC I/O requirements in the NERSC workload. IOR [1] was developed to set performance targets for LLNL's ASCI Purple system procurement. It focuses on measuring the sequential read/write performance under different file size, I/O transaction size, and concurrency. It also differentiates the strategies to use a shared file or one file per processor. More importantly, it supports both the traditional POSIX interface and the advanced parallel-I/O interfaces, including MPI-IO, HDF5, and parallelNetCDF. These alternative file strategies can be directly compared head-to-head for an identical set of testing parameters.

4. Organization of IOR benchmark

In this section, we describe the design of the IOR and its parameters. Figure 1 illustrates the relationship between the file structure and the processors when writing to a shared file.

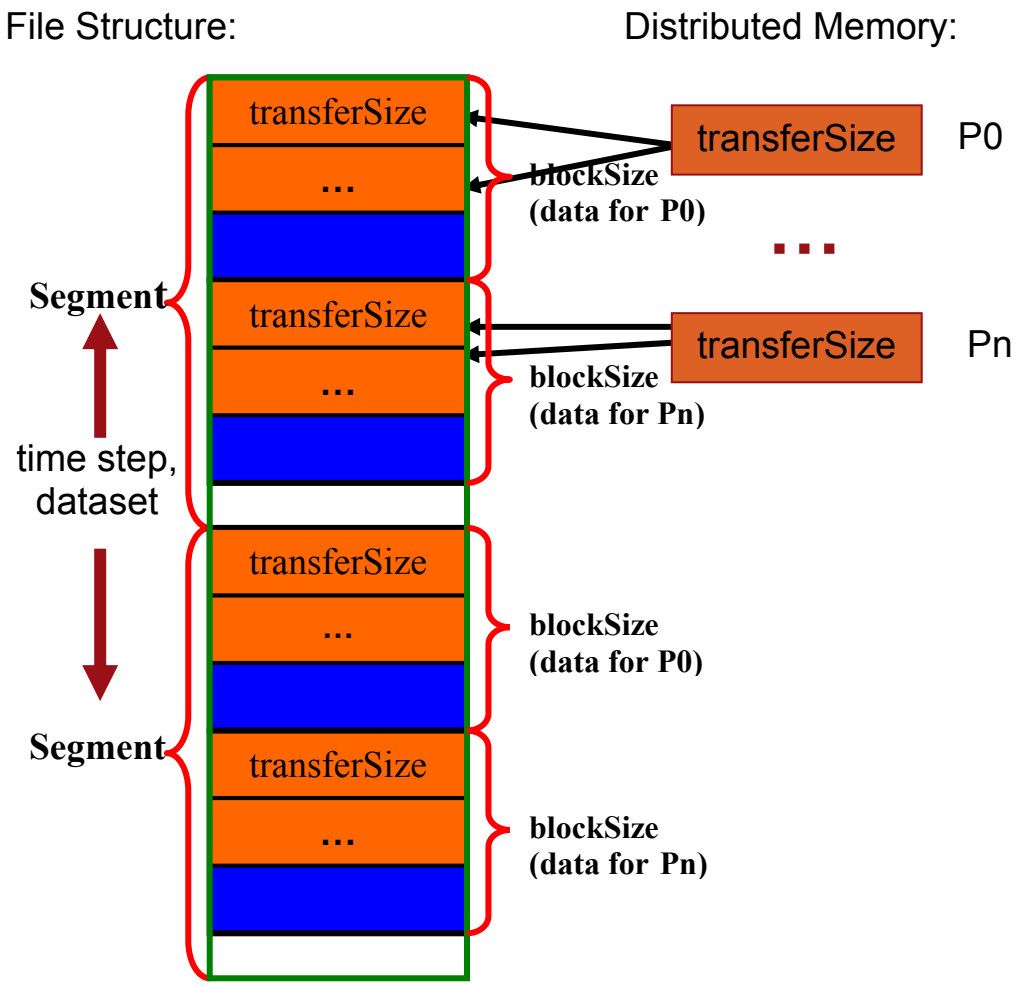

Fig. 1. The design of the IOR benchmark for shared file type. Blocks are stored in separate files for the 1-file-per-processor mode of operation.

Is organized as a sequence of "segments" that represent the application data for either one simulated time step of a single data variable (eg. pressure for timestep1, 2, 3, etc..) or a sequence of data variables (eg. pressure, temperature, velocity). For high-level file formats such as HDF5 and NetCDF, each segment directly corresponds to a "dataset" object in the nomenclature of these respective file formats. Each segment is divided evenly among the processors who share this data file into units called "blocks" to represent the array re-assembly performed by the parallel I/O layer. The process with rank 0 gets the first block and the process with rank 1 gets the second block and so on. The physical file layout corresponds to the application data resident in the distributed memories. Each block is further divided into many transfer units called TransferSize, in order to emulate the strided/stanza-like access patterns required to undo multi-dimensional domain decompositions, such as reassembling a bunch of 3D subdomains that reside on each processor into a single 3D logical array on disk. The TransferSize

chunks directly correspond to the I/O transaction size, which is the amount of data transferred from the processor's memory to file for each I/O function call (eg. the buffer size for a POSIX I/O call). For the one-file-per-processor case, the file structure is nearly identical to the diagram in Figure 1, but except that each process will write/read data to/from its own file (eg. each "block" is packed contiguously in separate files).

The following parameters of IOR are important to our study: *API, SegmentCount, BlockSize, FilePerProc, ReadFile, WriteFile, TransferSize, NumTasks*. The *API* describes which I/O API to use. Currently IOR supports POSIX, MPI-IO, HDF5, and NetCDF APIs. The *ReadFile* and *WriteFile* indicate whether the read operation or write operation will be measured. The *SegmentCount* decides the number of datasets in the file. The *BlockSize* represents the size of the subdomain of the dataset stored on each processor. The *TransferSize* is the I/O transaction size used to transfer data from memory to the data file, which may require multiple transfers per segment to copy the entire "BlockSize" to the data file. The *NumTasks* is the number of processors participated in the I/O operations.

5. I/O Performance Analysis

We selected two HPC platforms for comparison in our study. One is the IBM Power5/Fereration cluster running GPFS file system, located at NERSC. The other is the Cray XT4 from the Oak Ridge National Laboratory running luster file system. They represent two typical systems from HPC community. Given the relative size of these systems, it is important to compare the systems on the basis of performance characteristics rather than raw performance. Table 1 shows some of the highlights of these two architectures.

Table 1. The highlights of architectures and file systems of Bassi and Jaguar

Name	Location	File System	Processor	Interconnect	Peak I/O Band
Jaguar	ORNL	Lustre	Power5	SeaStart	42 GB/s
Bassi	NERSC	GPFS	Opteron	Federation	6.4GB/s

5.1. Platforms

Jaguar: Cray XT4 with Lustre

Jaguar the 11,701 node Cray XT4 supercomputer is located at Oak Ridge National Laboratory (ORNL) and utilizes the Lustre parallel filesystem. Each XT4 node contains a dual-core 2.6~GHz AMD Opteron processor, tightly-integrated to the XT4 interconnect via a Cray SeaStar ASIC through a 6.4~GB/s bidirectional HyperTransport interface. All the SeaStar routing chips are interconnected in a 3D torus topology, where each node has a direct link its six nearest neighbors. For the file system we tested, there are 144 OSTs, 18 DDN 9550 couplet, each delivering 2.3 ~ 3GB/s data transfer bandwidth, providing a theoretical 42~54GB/s aggregate I/O rate.

Bassi: IBM Power5 with GPFS
The 122-node Power5-based Bassi system is located at Lawrence Berkeley National Laboratory (LBNL) and employs GPFS as the global file system. Each node consists of 8-way 1.9 GHz Power5 processors, interconnected via the IBM HPS Federation switch at 4 GB/s peak (per node) bandwidth. The experiments conducted for our study were run under AIX~5.3 with GPFS v2.3.0.18. Bassi has 6 VSD servers, each providing sixteen 2~Gb/s FC links. The disk subsystem consists of 24 IBM DS4300 storage systems, each with forty-two 146 GB drives configured as 8 RAID-5 (4data+1parity) arrays, with 2 hot spares per DS4300. For fault tolerance, the DS4300 has dual controllers; each controller has dual FC ports. Bassi's maximum theoretical I/O bandwidth is 6.4 GB/s.

5.2 Caching Effects

When we measure the I/O performance, file caching (usually caused by the Unix "block buffer cache") result in anomalously high measured I/O rates because the data is being buffered in memory rather than hitting the disk. The block-buffer cache can use any unoccupied memory on a compute node to buffer I/O transactions and flush them to disk gradually in order to improve apparent I/O performance for small files. Therefore IO benchmarks must be scaled so as to exhaust the memory buffers and ensure that the performance of the underlying disk subsystem is actually being measured. In order to avoid caching effect on IO performance, benchmarks typically size the files to be several times larger than node memory size. We use a testing protocol that determines the optimal filesize to use to eliminate the influence of caching effects, by measuring the system's sensitivity to changes in the written file size. Ultimately we select a filesize for each system that demonstrates the least sensitivity to changes in filesize.

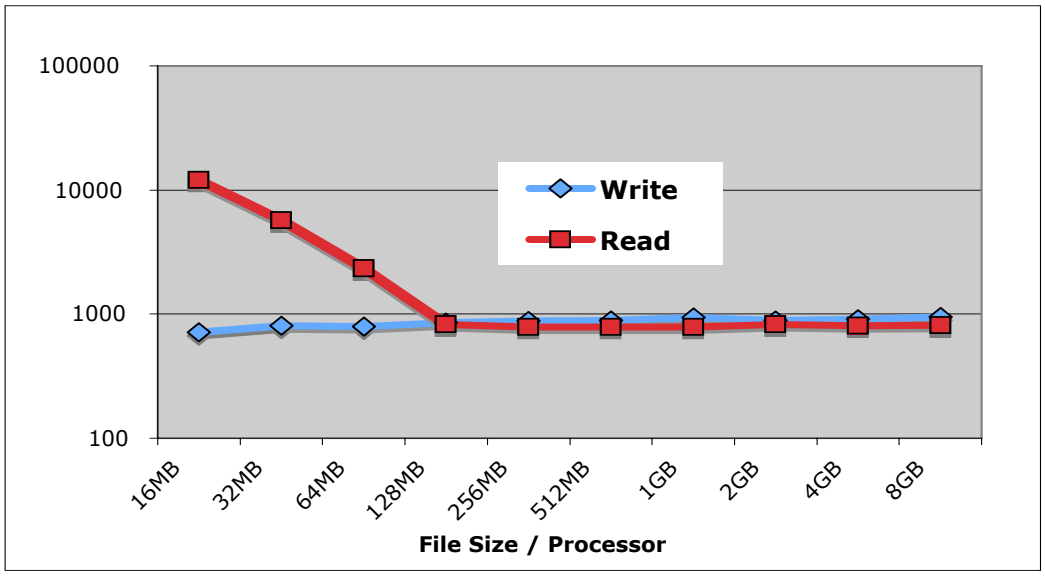

Fig. 2. The I/O performance under different file sizes on Bassi (using POSIX, one file per processor).

On bassi, the memory size is 32GB/node, i.e., 4GB/processor while on Jaguar, the memory size is 8GB/node, i.e., 4GB/processor. Fig. 2 shows the measured aggregate I/O bandwidth for a node using one file per processor strategy for different file sizes (by changing the blockSize in IOR to adjust file size, file size = blockSize*numTasks*segmentCount, the transferSize is fixed at 2MB and there is only one segment, 8 processors in a node on Bassi and 2 processors on Jaguar.)

When the file size is small, file caching has a considerable effect on the performance. We can clearly see the two performance regions on Bassi. When the file size is 16MB, the data is clearly being buffered in memory. At this time, the read performance is corresponds to the memory read performance, which is around12GB/s. With the increase of file size, the memory cache can no longer hold all the data and the read operation must get the data from the disks. The read performance degrades and gradually becomes stable when all data access is from disks. We see no caching effect on write. We find this behavior somewhat unexpected because many serial I/O systems, such as SGI's XFS exhibit a more pronounced write-caching than read-caching, where the apparent performance of writes is greatly exaggerated when the data files are smaller than memory.

Surprisingly, we see no virtually no caching effect on Jaguar (Fig. 3). There is a possibility that the Catamount microkernel on the compute nodes does not cache the file data on this specific system. The fact that the performance increases with the file size is probably an indication of metadata server overhead, which is gradually amortized by the larger file.

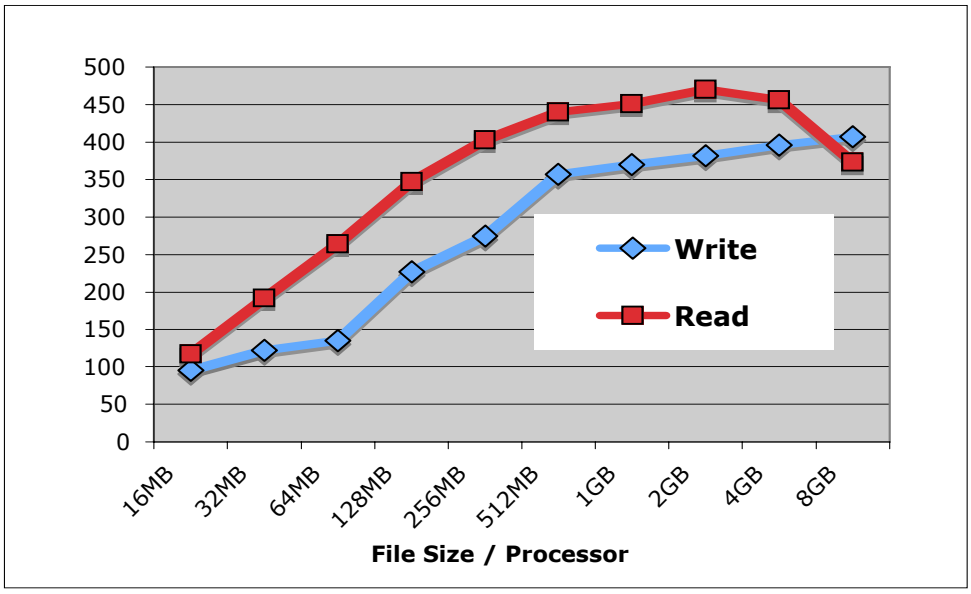

Fig. 3. The I/O performance under different file sizes on Jaguar (using POSIX, one file per processor).

By examining the results on these two platforms, we can notice that caching effects are platform dependent and can be significant on read performance. However, there is no a prior rule of thumb what file size to use to avoid the caching effect. To calibrate the filesize for our benchmarks, we used an exhaustive search for the filesize where first derivative of the performance was asymptotically zero (eg. least sensitive to changes in filesize). For the following measurements, we use 256MB and 2GB file size ***per processor*** for Bassi and Jaguar respectively.

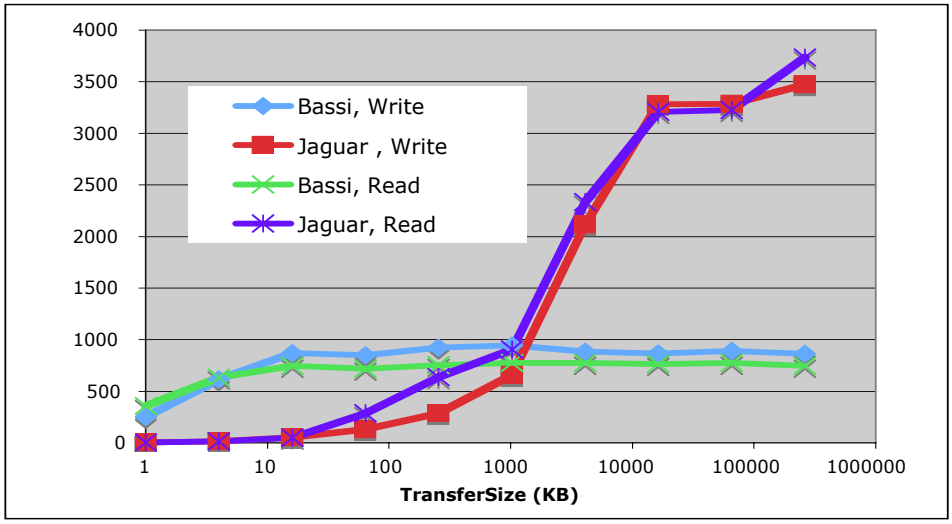

Fig. 4. The performance effect of transferSize on Bassi and Jaguar.

5.3 The effect of transaction size

The *transferSize* is the amount of data to be transferred each time for a processor between memory and file. Fig. 4 indicates that using larger *transferSize* is critical to achieve high I/O performance, especially on Jaguar. When *transferSize*=1KB, the system overhead is so high that only 2MB/s I/O performance is delivered. However, when 256MB *transferSize* is used, the I/O performance could go as high as 3500MB/s, amazingly 1700 times better. On Bassi, the performance gap between small and large *transferSize* is much more smaller. Jaguar is really focus on optimizing the performance for large files and large I/O transaction sizes.

5.4 The effect of Parallel IO

In the earlier section, we have discussed the advantage of using fewer files on large parallel platforms. Now, let's look at the performance difference in Fig. 5 (write performance) between using a single shared file by all processes and using a unique file by each processor. On Bassi, these two strategies perform similar to each other. On Jaguar, using shared strategy performs even better. This is perhaps related with the metadata server, which has to manage thousands of files for unique file case instead of one file for shared case. The read performance is very similar to the write performance.

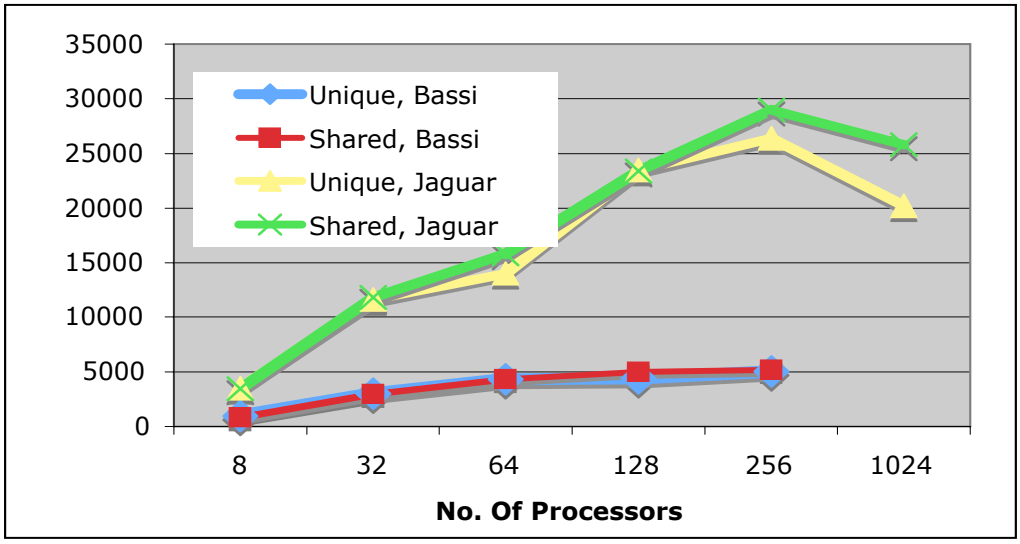

Fig. 5. The effect of file types on write performance on Bassi and Jaguar (transferSize= 2MB on Bassi, 256MB on Jaguar).

5.5 The effect of concurrency

Fig. 6 displays the aggregate read/write performance for different concurrencies in terms of MB/s using POSIX shared interface. (Note that given the POSIX shared file performance is nearly identical to the MPI-IO performance for these experiments, we present only the POSIX results for simplicity.) On Bassi, at first, the aggregate performance scales very well with increasing node counts (every node has 8 processors). After the number of processors reaches 64 (8 nodes), the improvement of performance starts to slow down and reaches its top when using 256 processors. We believe that the back-end of Bassi's GPFS storage system becomes saturated when the number of compute nodes (8) is modestly larger than the number of VSDs that connect to the disk subsystem (6 VSDs on Bassi).

On Jaguar, the performance scales well up to 256 processors and then starts to go degrade. Recall that althought this platform has over ten thousands nodes, the I/O peak is achieved at relatively low concurrency. But like Bassi, the peak is achieved when the number of compute nodes is modestly larger than the number of IO servers for the back-end of the disk subsystem. Also, there is a considerable performance gap between read and write performance for higher concurrencies.

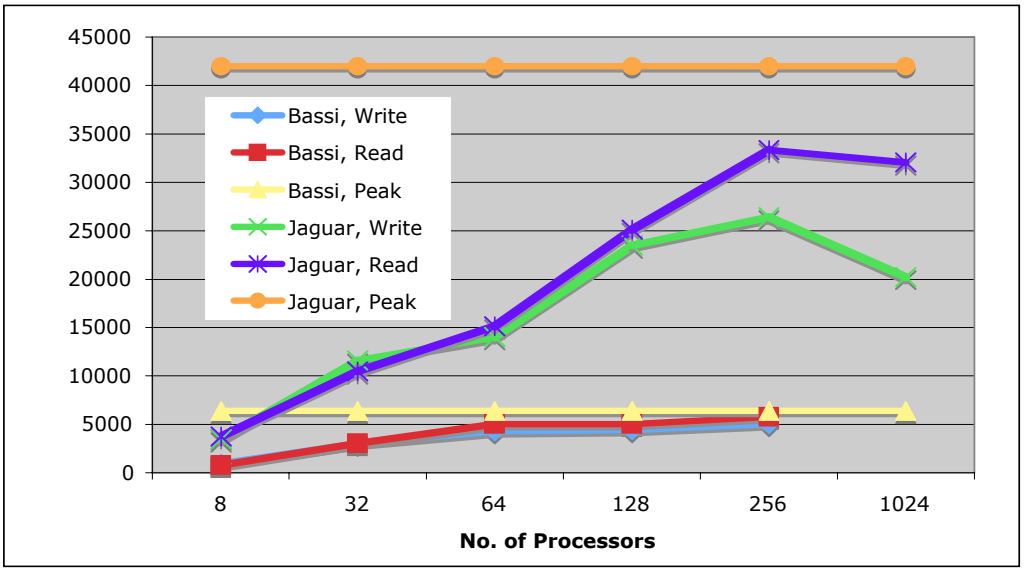

Fig. 6. The I/O scaling performance on Bassi and Jaguar for POSIX Interface

5.6 The effect of programming interfaces

Finally, let's look at the I/O performance under different programming interfaces. IOR provides us with the unique ability to directly compare multiple parallel IO strategies under identical testing parameters, for head-to-head comparisons under a variety of parametric conditions. Although the most recent release of parallelNetCDF is able to overcome the traditional 4Gigabyte filesize and dataset size limitations of the previous versions, we discovered that it still cannot accommodate dataset dimensions that are greater than 4billion elements[3]. Since the IOR benchmark expresses the datasets as 1D arrays we had to modify the current IOR to fold the arrays into additional (redundant) array dimensions. While this approach is impractical for realistic file storage purposes, it enabled direct comparisons of NetCDF performance some of the larger datasets that otherwise would not have been possible.

Fig. 7 and 8 exhibit the write performance for POSIX, MPI-IO, HDF5 (v1.6.5), and paralleNetCDF (v1.0.2pre). On Bassi, MPI-IO performs very similar to POSIX, followed closely by HDF5. However, the performance of parallelNetCDF falls far behind and performs worst. We have not been able to collect results for parallelNetCDF at higher concurrencies due to the long running time.

Jaguar shows similar performance behavior, although HDF5 performance is even more closely matched to the MPI-IO performance than on bassi. We note that the default striping on Lustre performs very poorly compared to one-file-per-processor. However, the user-level 'lstripe' command can be used to set the striping to maximum, 144 OSTs in the case of jaguar, which meets or even exceeds the one-file-per-processor performance.

The results show that users should expect performance of parallel IO strategies (concurrent access to a single file) that matches one-file-per-processor performance.

Furthermore, the results demonstrate that advanced file formats such as pHDF5 should be able to deliver performance comparable to that achieved when writing raw binary files using MPI-IO. This will be a great relief for many HPC users who worry that adopting the new parallel I/O interface may seriously slow down there applications.

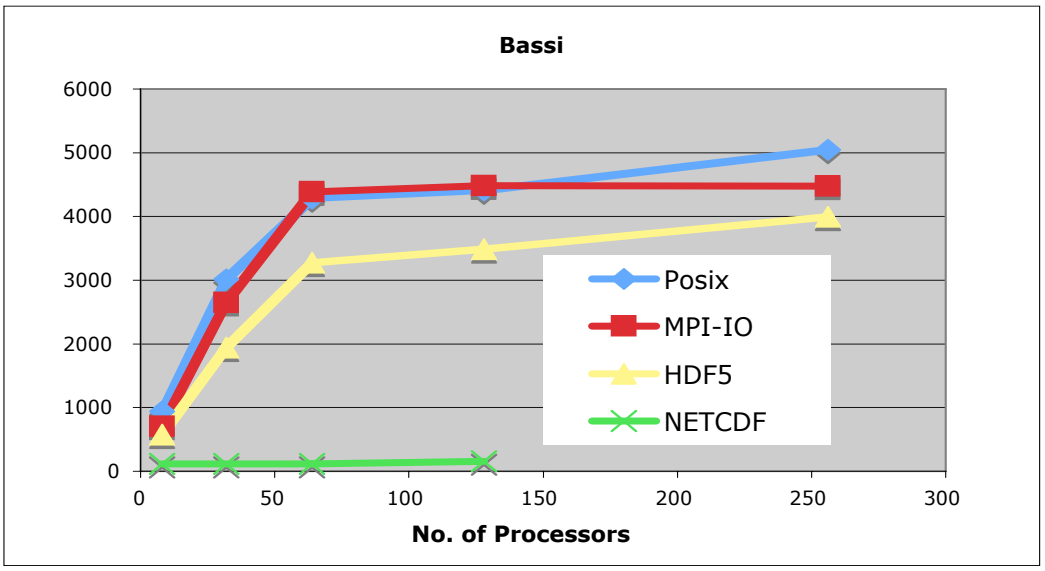

Fig. 7 The performance effect of different programming interfaces on Bassi.

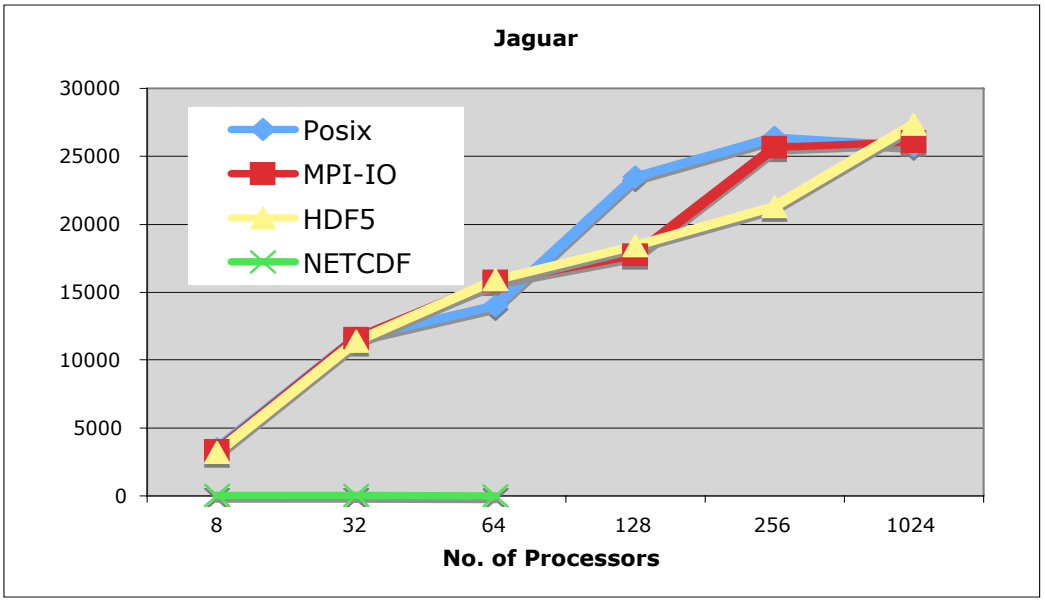

Fig. 8 The performance effect of different programming interfaces on Jaguar.

6 Application study

In this section, we are trying to relate the IOR performance to MADBench application performance (16). If successful, IOR could be used to mimic application I/O behavior.

MADBench is derived directly from the analysis of very massive Cosmic Microwave Background (CMB) datasets collected from satellite missions. Using a benchmark that is derived directly from the production scientific application allows us to study the architectural system performance under realistic I/O demands and communication patterns. The parameters that are closely related with I/O are:

- The size of the pseudo-data, nPixel. All the matrices have the size of nPixel*nPixel. Each matrix element is a double float variable.
- The number of the pseudo-data sets, nBin. There are total nBin matrices that are evenly distributed among all the participated processors (when nGang=1) or the subsets of the participants (when nGang > 1).
- The number of processor groups, Ngang. The processors are divided into Ngang groups so that each group is responsible to compute nBin/nGang matrix multiplications in the last phase of Madbench. The performance effect is a tradeoff between computation and communication.

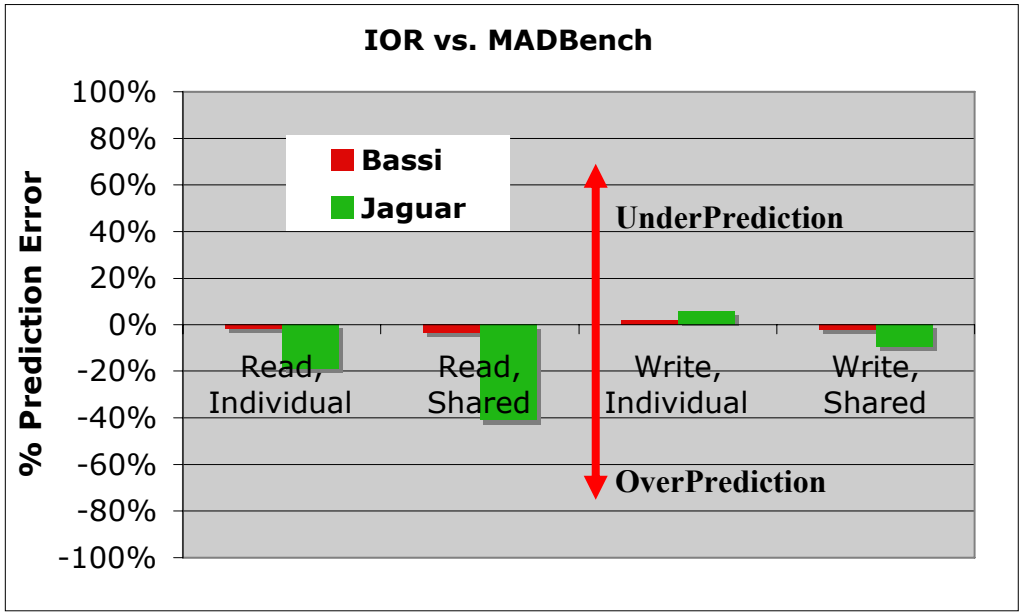

Fig. 9 The prediction error of using IOR to predict performance for MADBench.

Each matrix is written to the disk or read back independently. We assume nGang is 1 in our analysis. In this case, the memory buffer size on each processor is nPixel*nPixel*sizeof(double)/P. Madbench uses a weak scaling strategy, so the data set

size on a processor keeps constant. The typical memory buffer size used on a processor is 75Mb/s. The main I/O characteristic of this code all the processes read/write their local subsection of the dataset sequentially using large I/O buffer concurrently. Therefor, MadBench's IO pattern can be easily simulated using appropriate parameterization of the IOR benchmark.

Fig. 7 shows the prediction errors for 64 processors measured by IOR (transferSize=2MB, segmentCount=1, blockSize=76MB, API=MPI-IO) to predict MADBench I/O performance. We find that the predicted results match the MADBench results very well on Bassi. On Jaguar, the write performance also matches well. However, the read performance is modestly over-predicted. Currently we are examining whether this poor prediction for read performance on Jaguar is due to the interference of other applications during the benchmark data collection or system's interference or the read operation in MADBench is not implemented perfectly and needs to be improved. We are investigating other applications, including Chombo, FLASH, and GTC.

7 Summary

In this work, we analyzed the NERSC workload to develop a better understanding of the IO strategies used by a diverse array of applications. The workload analysis led to the selection of IOR as our synthetic benchmark to represent the requirements of the NERSC workload. We used IOR to study the I/O performance of two popular HPC platforms.

We have shown that the I/O performance of current HPC systems is highly affected access file type, access pattern, file size, I/O transaction size (transfer size), and concurrency. On Jaguar, in order to obtain good performance, large file size and large I/O transaction size are absolutely required. On Bassi, such requirements are not quite as demanding. We find that users should expect parallel IO and even advanced parallel file formats such as HDF5 to match the performance of more primitive one-file-per-processor POSIX IO. Rumors of performance loss that have discouraged adoption of advanced file formats are potentially exaggerated, but the performance of parallelNetCDF falls far behind the other interfaces.

We also observe that the best performance is achieved at relatively low concurrency on both systems, so it is important not to assess parallel IO "scalability" in terms of parallel speedup that exactly matches the speedup in FLOPs. One must determine what concurrency is required to reach saturation for the disk subsystem and set expectations based on that metric. IOR can be used to set expectations as to when the IO subsystem achieves saturation.

Finally, we show that by choosing IOR parameters, it can be used to mimic the access patterns of real applications. Not only does it mimic the access patterns, it is also relatively good at predicting the performance for these applications. Further application studies will appear in upcoming papers.

References

[1] IOR, http://www.llnl.gov/asci/purple/benchmarks/limited/ior/.

[2] Iostone, http://www.iostone.org

[3] Bonnie, http://textuality.com/bonnie

[4] PRIOmark, http://www.ipacs-benchmark.org/index.php?s=download&unterseite=priomark

[5] Input/Output IObench, http://www.sdsc.edu/pmac/Benchmark/iobench/

[6] Effective I/O Bandwidth Benchmark, http://www.hlrs.de/organization/par/services/models/mpi/b_eff_io/

[7] ParallelNETCDF, http://trac.mcs.anl.gov/projects/parallel-netcdf

[8] http://www.opengroup.org/platform/hecewg/uploads/40/10894/POSIX-extensions-some-goals.pdf

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

[10] HDF5, http://hdf.ncsa.uiuc.edu/HDF5

[11] J. Li, W.K. Liao, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, and M. Zingale, "Parallel netCDF: A High-Performance Scientific I/O Interface", SC2003.

[12] Peter M. Chen, David A. Patterson, "A new approach to I/O Performance Evaluation Self-Scaling Benchmarks, Predicted I/O Performance", Proceedings of the 1993 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems.

[13] ASC/Alliance Center for Astrophysical Thermonuclear Flashes, http://flash.uchicago.edu/website/home/

[14] MADbench2, https://crd.lbl.gov/~borrill/MADbench2/

[15] K. Antypas, A.C.Calder, A. Dubey, R. Fisher, M.K. Ganapathy, J.B. Gallagher, L.B. Reid, K. Reid, K. Riley, D. Sheeler, N. Taylor, "Scientific Applications on the Massively Parallel BG/L Machines", http://ww1.ucmss.com/books/LFS/CSREA2006/PDP4125.pdf