Scalable I/O and analytics

Alok Choudhary[1], Wei-keng Liao[1], Kui Gao[1], Arifa Nisar[1], Robert Ross[2], Rajeev Thakur[2], and Robert Latham[2]

[1] Electrical Engineering and Computer Science Department, Northwestern University
[2] Mathematics and Computer Science Division, Argonne National Laboratory

E-mail: {choudhar,wkliao,kgao,ani662}@eecs.northwestern.edu,
{rross,thakur,robl}@mcs.anl.gpv

Abstract. High-performance computing systems have already approached peta-scale with hundreds of thousands of processors/cores in many deployments. These systems promise a new level of predictive and knowledge discovery ability as researchers gain the capability to model dependencies between phenomena at scales not seen earlier. These applications are highly I/O and data intensive, leading scientists to observe that performing I/O and subsequent analyses are major bottlenecks in effectively utilizing peta-scale systems and a major hurdle in accelerating discoveries. Although significant progress has been made in performance, interfaces, and middleware runtime systems for I/O in the recent past, significantly more research and development needs to be carried out to scale the performance to the desired levels for systems containing tens to hundreds of thousands of cores. In this work we outline our recent achievements and current research for designing scalable I/O software and enabling data analytics in storage systems. We also enumerate key challenges for the I/O systems and discuss ongoing efforts that address these challenges.

1. Introduction

Research and development in science and engineering is increasingly based on simulations and/or on analysis of observational data requiring the use of high-performance computing (HPC) systems. HPC systems have already approached peta-scale in many deployments and efforts are already underway towards designing and scaling systems to exascale. As systems scale, simulations in areas such as combustion, chemistry, nanoscience, astrophysics, cosmology, fusion, climate prediction, environmental science, and biology have the potential of providing a new level of predictive and knowledge discovery, as researchers gain the capability to model dependencies between phenomena at scales. Many of these applications are expected to use systems with hundreds of thousands of processors/cores in order to achieve the resolution needed to better analyze and understand complex science problems. One common characteristic of all these applications is that they are all highly I/O and data intensive.

For example, in the area of atmospheric science and climate modeling it would be possible to use more realistic models of clouds, precipitation, and convective processes by substantially increasing the grid resolution, which would be feasible with computing power at this scale. At a mesh spacing of 1 km and higher resolution, it is possible to accurately predict tropical cyclones and other extreme weather events. It has been estimated that a global model on a uniform grid with 1 km mesh spacing would require a computer with sustained performance of 10 peta flops or more [1]. These types of applications require simulations, data assimilation,

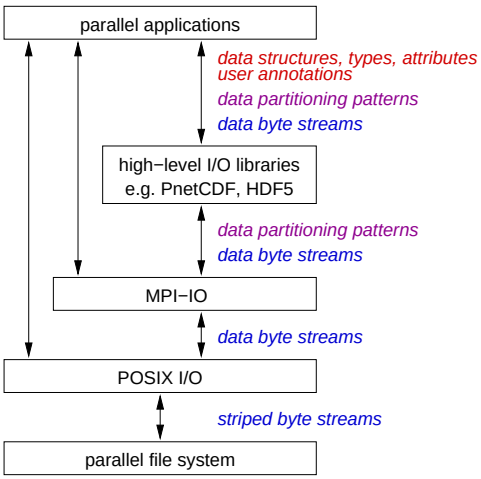

Figure 1. I/O software stack. High-level data information is lost as data reaches lower levels.

analytics, validation of simulations with observations, etc. This requirement is quite common across a range of applications in science and engineering. However, these characteristics across applications also lead the scientists to observe that performing I/O and subsequent analyses are major bottlenecks and most scientists also point that out as a major hurdle to effectively utilizing peta-scale systems and accelerating discoveries.

Although significant progress has been made in performance, interfaces, and middleware runtime systems for I/O in the recent past, significantly more research and development needs to be carried out to scale the performance to the desired levels for systems containing tens to hundreds of thousands of cores. Figure 1 illustrates a typical parallel I/O software stack that is found in typical HPC systems. We note that this is a logical view and not meant to illustrate physical connectivity or configuration of the I/O stack, rather to describe the software layers. A parallel application may directly call MPI-IO or POSIX I/O functions to access files from the parallel file system. Alternatively, a typical large-scale data-intensive application may use several layers of software as illustrated in the figure. For example, a weather modeling application may use NetCDF [2] for data representation and storage, where I/O is carried out through parallel netCDF (PnetCDF) [3] for fast performance. PnetCDF is built on top of MPI-IO [4], which may be layered on top of the Parallel Virtual File System (PVFS) [5] or some other parallel file system. The highest layer normally has semantic information associated with the data, such as data structures, types, attributes, user annotations, and data partitioning patterns, which describe the intent of a user. For example, at the high-level I/O library layer, the user may have represented a three-dimensional structure with a specific distribution of data (each element of which may be a collection of variables such as temperature, pressure, density, humidity, etc.). Some relationships can be preserved in the MPI-IO layer, such as the data partitioning patterns among a group of processes. Furthermore, the MPI-IO may use the partitioning information to enable optimizations of data caching, process coordination, and other techniques for performance optimizations [6, 7, 8, 9, 10, 11]. However, the data structures, types, attributes, and annotations will be converted to byte streams at the MPI-IO layer. Further down, at the POSIX I/O and file system layers, without proper interfaces for complex metadata communication, high-level information gets lost, and data appears just a stream of bytes.

As we move towards peta-scale systems and beyond, we have to reconsider the traditional approaches to I/O optimizations. Traditional interfaces in file systems and storage systems are designed to handle the worst-case scenarios for conflicts, synchronization, locking, coherence checks, and other issues, which adversely affect the performance. In other words, in this

conventional design, the lower layers of the software stack are designed pessimistically, rather than optimistically. Of course, the reasons for those designs are the semantics and expectations of consistency, reliability, and other features needed from the storage and file systems. However, by providing and using the appropriate interface at the middleware (e.g., MPI-IO) level, understanding the access patterns, using some system resources within user application space, and performing intelligent caching, the desired scalability in performance can be achieved in such a manner that the optimizations are also portable across different systems through the use of this middleware. We need to understand the source of the problems and limitations of the current approaches, rather than blindly trying to deal with their manifestations for particular architectures and applications. In many cases, the problem is not that of having insufficient I/O capacity or bandwidth, but it is the excessive synchronization and file system control overhead at the I/O layer to account for semantics and the worst-case scenarios. This can have devastating results on overall system performance. Even though research and development for implementing efficient memory synchronization or file locking scheme (when really needed) is important, high-level information about access patterns may be the keys to eliminate or reduce such bottlenecks and are, therefore, critical to achieving real scalability. In Sections 2 and 3, we discuss our recent research on developing I/O strategies for PnetCDF and MPI-IO, respectively.

The storage systems in most of today's HPC computers employ a group of servers dedicated to I/O tasks and managed by a parallel file system. The computational capacity of one of these I/O servers is equal or even greater than that of one of the compute nodes that run the application programs. However, due to little computation required for serving I/O requests, the CPUs at the I/O servers are underutilized. The computational resources at the storage systems have been proposed as active storage to perform data filtering, a common operation in database queries. Examples are active disks that embed processing hardware into disk devices so that data filtering can be accomplished much faster. Utilizing the computational resources on the I/O servers for data processing extends the same idea of reducing the data amount transferred between application clients and the storage system. There are two advantages of enabling data processing at I/O servers over active disks. First, the I/O servers require no special hardware, unlike active disks which are still not in mass production. Second, the I/O servers are interconnected and can potentially serve as a computer cluster, running full-blown parallel programs.

Computational scientists must understand results from experimental, observational and computational simulation data to gain insights and achieve knowledge discovery. As systems approach the petascale range, problems that were unimaginable a few years ago are within reach. With the increasing volume and complexity of data produced by ultra-scale simulations and high-throughput experiments, understanding the science is largely hampered by the lack of comprehensive I/O, storage, data manipulation, analysis, and mining tools. Scientists require techniques, tools, and infrastructure to facilitate better understanding of their data, in particular the ability to effectively perform complex data analysis, statistical analysis and knowledge discovery. In Section 4, we discuss the issues of enabling the storage systems to carry out data analytics computation as part of I/O operations. Without the proper interfaces and standardized protocols, data structures and their access information at the higer levels of the I/O stack cannot be passed to the lower leverls systematically.

2. Optimizations in parallel netCDF

Dataset storage, exchange, and access play a critical role in scientific applications. For such purposes, netCDF serves as a software library and self-describing machine-independent data format that supports the creation, access, and sharing of array-oriented scientific data [2]. NetCDF stores metadata in a file header that describes the structures of the arrays and their layout in a netCDF file. Additional information, such as user annotations, can also be saved as

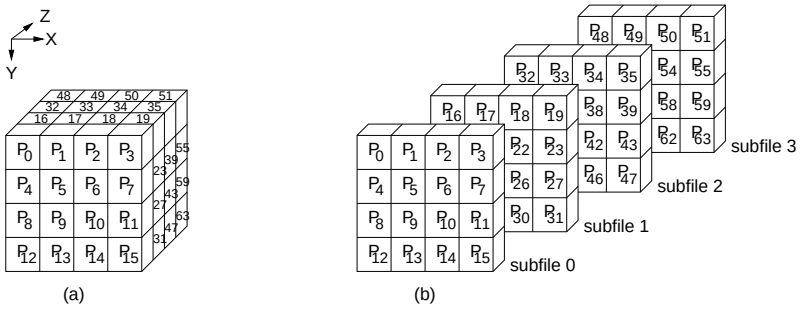

Figure 2. A 3D array partitioned among 64 processes and its mapping to 4 subfiles.

attributes. Built on top of MPI-IO for portability, Parallel netCDF (PnetCDF) was developed to support parallel I/O operations while retaining the netCDF file format for compatibility with existing netCDF datasets and tools [3]. PnetCDF I/O functions take an additional argument of an MPI communicator to indicate the processes participating in the shared-file I/O operations. PnetCDF internally constructs MPI derived data types to describe complex access patterns, such as reading a subarray of a multidimensional variable. MPI hints supplied by users are passed to the underlying MPI-IO library, so that PnetCDF can take advantage of the I/O optimizations available in MPI-IO.

PnetCDF encourages a shared-file I/O approach, one of the two parallel I/O programming styles commonly used by today's parallel applications, where all application processes store and access related data in single, shared files. The other style is termed as unique I/O, also known as the unique-file-per-process style, in which each process accesses files that are unique to the process. Unique I/O can create a management nightmare for file systems when a parallel job running on thousands of processes produces hundreds of thousands or millions of files. Furthermore, accessing millions of files can be a daunting task for post-run data analysis.

The shared-file I/O programming style avoids such problems. It can also be used to save global partitioned data structures in the canonical order. However, shared-file I/O performance often suffers from significant file system overheads, at large scale, due to conflicted I/O requests among processes [12, 11]. Such overheads include the system's data consistency control for cache coherency and I/O atomicity. Standard file system atomicity rules require that the results of an individual write call are either entirely visible or completely invisible to any successive read call from the same or different process [13]. Similarly, any caches must be kept synchronized so that a consistent view of the file is preserved. In modern parallel file systems, data consistency control is usually implemented using a file locking mechanism. As the number of concurrent accesses to a shared file increases, the overhead due to lock conflicts can increase dramatically.

As a middle ground between the two extremes of small files and very large files, we have developed a subfiling scheme for PnetCDF that allows a large multi-dimensional global array to be split into a number of smaller subarrays, each saved in a separate file. The subfiling scheme reduces the file system control overhead by decreasing the number of processes concurrently accessing a shared file. By saving the subfiling information in the netCDF files, the structure of global arrays can be portably reconstructed. In PnetCDF subfiling, the multi-dimensional array partitioning is always along the most significant axis, and subarrays are stored in the canonical order. Figure 2 shows an example of a three-dimensional array partitioned among 64 processes and the mapping of the subarrays to four subfiles. Through the same PnetCDF I/O functions, the partitioned global array still appears to users as a single netCDF array and access to it is kept the same as without subfiling. In order to achieve this goal, the subfiling implementation automatically generates the MPI fileviews for each subfile for a given I/O request. Mapping information of a global array to subfiles is stored as netCDF attributes which are duplicated

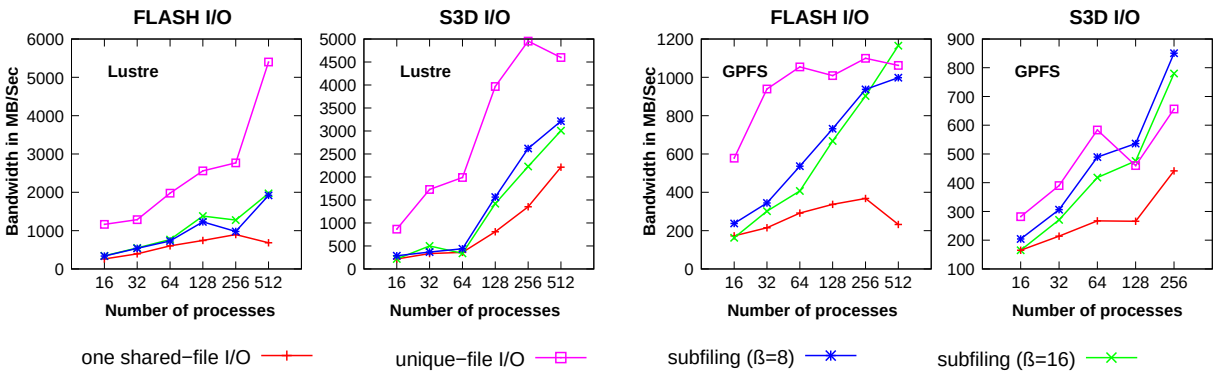

Figure 3. PnetCDF subfiling performance results for FLASH and S3D I/O kernels.

in all subfiles. Each netCDF subfile is self-contained with sufficient information to describe the layout of the local array mapped to the global array, including the names of all subfiles and partitioning of each subfile. Therefore, accessing the metadata from one of the subfiles is enough to understand the subfiling structure of the global array.

We evaluate this subfiling scheme using two scientific production application I/O kernels, FLASH-IO and S3D-IO. The performance is compared with the approaches of unique I/O and single shared-file I/O using two parallel file systems: Lustre on Franklin, the Cray XT4 at NERSC, and GPFS on Mercury, a TeraGrid machine at NCSA. Figure 3 summarizes the write bandwidths where β = nproc/nsubfile is the ratio of the number of processes sharing a subfile. In most of the cases, subfiling performance falls between the two I/O approaches. On GPFS, subfiling even outperforms the unique-file I/O approach when 256 or 512 numbers of processes are used. This is because the large number of concurrent file open operations overwhelms the file system metadata server, causing significant increase in the file open cost.

3. Scalable MPI-IO

The storage systems on high-performance parallel computers usually consist of a small number of I/O servers to serve requests from a much larger number of compute nodes that run user applications. As the size of modern computer systems increases, it is obvious that the storage system is becoming a performance bottleneck. Envisioning this problem, the IBM BlueGene systems dedicate a set of nodes as intermediate I/O nodes that forward the requests from application nodes to the I/O servers. These I/O nodes become the actual I/O clients to the underlying file system, where one I/O node is responsible for the requests from a group of compute nodes [14]. For example, in BG/L, there is usually an I/O node for every 8 to 64 processors [15, 16]. This design decreases the chances of possible I/O server contention, when multiple compute nodes are concurrently performing I/O. Imagine, if thousands of process clients on a HPC system perform I/O operations simultaneously, then the high volume of I/O requests can easily choke the available network bandwidth at the storage system.

Based on a similar concept, we have developed a software solution, the I/O Delegate Cache system (IODC), that uses an additional set of MPI processes as I/O delegate nodes (IOD) dedicated to handle I/O requests from application client processes [17]. The IODC system is built inside ROMIO [18], an MPI-IO implementation developed at Argonne National Laboratory. In the IODC system, all I/O requests initiated by a user application running on application nodes are intercepted and redirected to IOD nodes. This is achieved when parallel applications pass their I/O requests to the MPI-IO layer directly or through a high-level I/O library, e.g., HDF5 [19] or PnetCDF. Like the I/O nodes in a Blue Gene, only IOD nodes access the underlying parallel file system directly. In addition to forwarding requests, a collaborative distributed

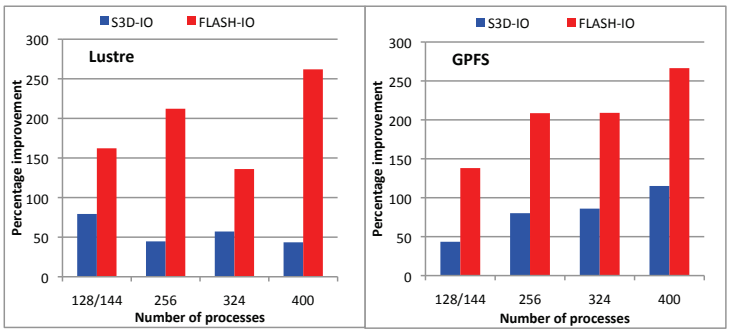

Figure 4. Performance results of using I/O delegation for FLASH and S3D I/O kernels.

cache is maintained across the IOD nodes for further I/O optimization. Features such as request aggregation, lock request alignment, cache page migration, and atomicity control are incorporated. By exploiting processing power and memory of IOD nodes for performing I/O caching and data aggregation, we achieve significant I/O bandwidth improvements.

We conducted our experiments using the FLASH and S3D I/O kernels using Lustre on Tungsten and GPFS on Mercury. Both Tungsten and Mercury are TeraGrid machines at NCSA. Experimental results are given in Figure 4. By allocating only 2-3% of additional computer nodes as I/O delegates, we achieved the improvement percentages ranging from 25% to 260%. Such significant performance improvement demonstrates the potential for using a separate group of MPI processes for I/O tasks to reduce the contention at the I/O servers, a technique that is applicable to both shared-file and unique-file I/O patterns.

3.1. Adaptive file domain assignment for collective I/O

Although the shared-file I/O method provides a way to preserve the canonical order of structured data and ease the file management for post-run data analysis and visualization, this method often performs poorly when the I/O requests are not well coordinated. To address such performance issue, MPI collective I/O is designed to provide an opportunity to induce collaboration among the requesting processes for better I/O performance. Well-known examples of using such a collaboration are two-phase I/O [20] and disk directed I/O [21]. These process collaboration strategies have demonstrated significant performance improvements over uncoordinated I/O. However, even with these improvements, the shared-file can I/O still perform far worse than the unique-file-per-process approach. One of the main reasons is that concurrently accessing shared files incurs higher file system locking overhead from data consistency control. The same overhead can never happen if a file is only accessed by a single process. Two of most important data consistency issues that POSIX-compliant file systems must enforce are I/O atomicity and cache coherence. Most file systems rely on a locking mechanism for these tasks. Due to the nature of file striping, the lock granularity is usually set to the file block size or stripe size, instead of a byte. If two I/O requests simultaneously access the same file block and at least one of them is a write, they must be carried out serially, even if they do not overlap in bytes.

ROMIO's collective I/O implementation is based on the two-phase I/O strategy proposed in [20], which includes a request redistribution phase and an I/O phase. The two-phase strategy first calculates the aggregate access file region and then partitions it among the I/O aggregators into approximately equal-sized contiguous segments, named **file domains**. This partitioning strategy is referred as the even partitioning method. The I/O aggregators are a subset of the processes that act as I/O proxies for the rest of the processes. In the data redistribution phase, all processes exchange data with the aggregators based on the calculated file domains. In the I/O phase, aggregators access the shared file within the assigned file domains. Two-phase I/O can

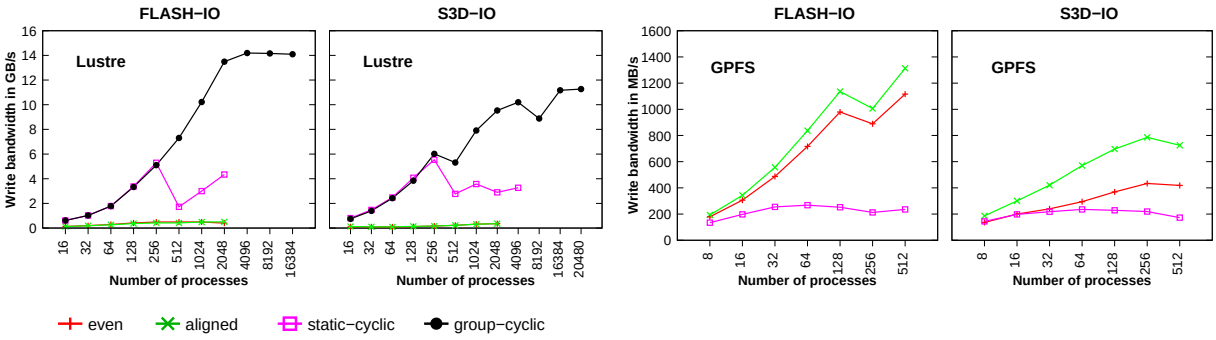

Figure 5. Results for various file domain partitioning methods for FLASH and S3D I/O kernels.

combine multiple small non-contiguous requests into large contiguous ones and has demonstrated to be very successful, as modern file systems handle large contiguous requests more efficiently. However, the even partitioning method does not necessarily produce the best I/O performance on all file systems due to the lock contentions. Since the lock granularity on most parallel file systems is set to the file stripe size, the file domain partitioning must be carefully aligned with the lock boundaries in order to minimize the lock conflicts.

We have developed a few file domain partitioning methods [22] as replacements for the even partitioning method used by ROMIO. The first method aligns the partitioning with the file system's lock boundaries. The second method, the *static-cyclic* method, partitions a file into fixed-size blocks based on the lock granularity and statically assigns the blocks in a round-robin fashion among the I/O aggregators. The third method, the *group-cyclic* method, divides the I/O aggregators into groups, each being of size equal to the number of I/O servers. Within each group, the static-cyclic partitioning method is used. This method is particularly designed for the situation that the number of I/O aggregators is much larger than the I/O servers. We evaluate these methods using four I/O benchmarks on two parallel file systems, Lustre and GPFS. Due to the different file locking mechanism implemented in Lustre and GPFS, these partitioning methods result in significant performance differences on the two file systems.

Figure 5 presents the write bandwidth results of the FLASH-IO and S3D-IO benchmarks on Jaguar, a Cray XT4 at ORNL, and Mercury at NCSA. These results indicate that the group-static and lock-boundary aligned methods give the best write performance on Lustre and GPFS, respectively. The lessons learned from this work can be helpful for the MPI-IO implementation to adapt the best partitioning method based on the underlying file system configuration. On file systems that implement server-based locking protocols, such as the Lustre, the group-cyclic file domain partitioning method is the best choice for collective write operations, as it minimizes the lock acquisition cost. For token-based locking protocols, such as the one used by GPFS, the method that aligns the partitioning to the lock boundaries produces the least number of file lock conflicts. These observations support developing ad hoc file domain partitioning methods based on the underlying file system characteristics, such as file locking and striping.

4. Data analytics in storage systems

It is known that most large-scale applications carry out their I/O tasks in bursts. That is, periodically a significant time of I/O inactivity is followed by bursts of I/O. Figure 6 illustrates the I/O activities of the CTH application in one 3-hour run. CTH is a family of codes developed at Sandia National Laboratories for modeling complex multi-dimensional, multi-material problems that are characterized by large deformations and/or strong shocks [23, 24]. Through our collaboration with the Sandia National Laboratory, we are studying the I/O traces produced from production runs [25]. Data checkpointing was enabled for every 25 minutes and

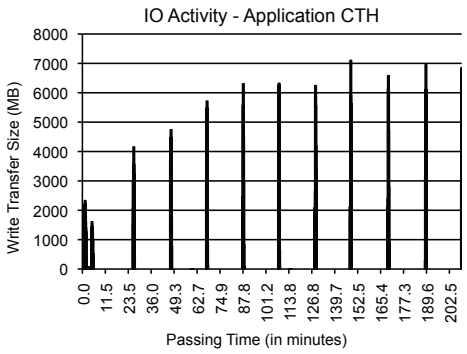

Figure 6. The bursty nature I/O activity of application CTH. Checkpoint writes occurs at a regular interval. This I/O pattern is very common in large-scale applications.

hence results in bursts of I/O activities on the file system. Such I/O pattern explains a low utilization of I/O servers commonly observed in today's parallel file systems.

Active storage has been proposed since late 1990s and prototyped in hard disk drives [26, 27, 28]. The majority of active storage applications are database operations, which perform mostly data filtering operations. The concept of active storage has been extended from the disk devices to the entire storage system, particularly, the I/O servers [29, 30]. Given the computational capabilities of the I/O servers in a parallel file system, their idle time can potentially be used to share the burden of computational nodes. Data intensive operations, such as data analytics and mining, can benefit from such as an active storage model on parallel file systems. However, offloading computational tasks from application clients to file system servers has several obstacles on today's file systems.

As illustrated in Figure 1, high-level information is lost when the data arrives at the file system. Information such as data types and structures are critical for data analytics programs to operate correctly. On the POSIX-compliant parallel file systems, the file metadata that can be transferred between client applications and file systems is limited to the attributes describing the file status. To enable computation offloading from application clients to file systems, the file system must adapt object-based protocols to allow user-defined attributes and operations be associated with the data objects. Object-based concepts have been incorporated into some parallel file systems, such as PVFS, Lustre, and Panasas, but without standardized metadata protocols, the potential optimization from metadata availability at the file level is still difficult to achieve.

Another challenge is the collaboration of multiple file system servers in order to carry out a single data analytics task in parallel. As files stored on a parallel file system can be striped across multiple servers, these servers must be able to communicate with each other as part of a parallel data analytics task. However, the I/O servers in modern parallel file systems are not programmed to communicate with each other directly. Enabling active storage on parallel file systems will soon change this phenomenon. Offloading the analytics tasks from application clients to the I/O servers will require identifying the group of servers holding parts of the data and informing the server group of data partitioning patterns. This is analogous to defining an MPI communicator in an MPI program, carried out at run time.

We have started the active storage design for parallel data analytics by extending the PnetCDF library and the PVFS file system. The well defined metadata in PnetCDF is borrowed in order to construct the interfaces that allow users to pass customized metadata along with the data to the PVFS file system. PVFS servers collaborate to execute data analytics operations as parallel programs. Our initial work also uses the I/O delegation discussed in Section 3 as

another option for operation offloading. Since I/O delegation runs at user space, close to the applications, it can retain more high-level metadata details for better performance. In our design, the results of analytics are represented and saved in the netCDF format, so that both client and server components have a consistent and metadata-rich data presentation. Results are forthcoming.

5. Conclusions

Hardware and software improvements are constantly being made in the area of HPC I/O to address the needs of efficient and effective storage system in computational science. However, there are still challenges for achieving scalable performance as HPC systems enter the peta-scale era. The most significant one is obsolete standards that cannot address large-scale I/O requirements. From the performance point of view, abiding by the traditional semantics for data consistency has been identified as a major obstacle. Such semantics were designed for small-scale, loosely-coupled distributed environments and the problems addressed by these semantics seldom appear in parallel I/O operations. New standards for interfaces and semantics must be adopted to enable performance at scale. From the productivity point of view, the high-level I/O information has no proper communication channels to be delivered to other components in the I/O stack. Many aspects of such information can be used to bypass unnecessary system controls and enable optimizations for better performance. Our work demonstrated the importance of reducing file system locking overhead in collective MPI-IO operations. Such optimization relies on the awareness of application I/O patterns as well as the underlying system configurations. Extensions to POSIX I/O, the ANSI T-10 OSD, and pNFS standards are all important steps in the direction of high-performance, standards-based I/O systems.

To increase the functionality of I/O systems for computational science, flexible interfaces are important for exploiting the computational potential in storage systems. We have highlighted design issues for enabling data analytics in parallel file system servers and pointed out the deficiency of current standards. As the I/O systems and software are still evolving to meet the challenges of existing and future HPC systems, adapting informative application programming interfaces in file systems will elevate the I/O systems to a more versatile component from its traditional role of simple file access.

Acknowledgments

This work was supported in part by U.S. Department of Energy (DOE) SCIDAC-2: Scientific Data Management Center for Enabling Technologies grant DE-FC02-07ER25808, DOE FASTOS award number DE-FG02-08ER25848, the DOE Office of Advanced Scientific Computing Research, Office of Science, under contract DE-AC02-06CH11357 and DE-FG02-08ER25835. This work was also supported in part by the National Science Foundation (NSF) under HECURA CCF-0621443, NSF SDCI OCI-0724599, and NSF ST-HEC CCF-0444405. We thank Lee Ward and Ruth Klundt at Sandia National Laboratories for providing the I/O trace data and valuable suggestions toward the application I/O characteristic analysis on HPC systems. This research was supported in part by NSF through TeraGrid resources provided by NCSA, under TeraGrid Projects TG-CCR060017T, TG-CCR080019T, and TGASC080050N. This research also used resources of the National Center for Computational Sciences at Oak Ridge National Laboratory, which is supported by the Office of Science of DOE under Contract No. DE-AC05-00OR22725.

References

[1] Committee 2008 *National Research Council Report, National Academies Press: Washington, D.C.*

[2] netCDF *Network Common Data Form* The UniData program center, University Corporation for Atmospheric Research http://www.unidata.ucar.edu/software/netcdf/

[3] Li J, Liao W, Choudhary A, Ross R, Thakur R, Gropp W, Latham R, Siegel A, Gallagher B and Zingale M 2003 Parallel netCDF: A Scientific High-Performance I/O Interface *the Supercomputing Conference*

[4] Message Passing Interface Forum 1997 *MPI-2: Extensions to the Message Passing Interface* http://www.mpi-forum.org/docs/docs.html
[5] Carns P, Ligon W, Ross R and Thakur R 2000 PVFS: A Parallel File System for Linux Clusters *the Third Annual Linux Showcase and Conference* pp 317–327
[6] Ching A, Choudhary A, Coloma K, Liao W, Ross R and Gropp W 2003 Noncontiguous I/O Accesses Through MPI-IO *IEEE/ACM International Symposium on Cluster Computing and the Grid*
[7] Ma X, Winslett M, Lee J and Yu S 2003 Improving MPI-IO Output Performance with Active Buffering Plus Threads *the International Parallel and Distributed Processing Symposium*
[8] Coloma K, Choudhary A, Liao W, Lee W, Russell E and Pundit N 2004 Scalable High-level Caching for Parallel I/O *the International Parallel and Distributed Processing Symposium*
[9] Coloma K, Choudhary A, Liao W, Lee W and Tideman S 2005 DACHe: Direct Access Cache System for Parallel I/O *the 20th International Supercomputer Conference*
[10] Coloma K, Ching A, Choudhary A, Liao W, Ross R, Thakur R and Ward L 2006 A new flexible MPI collective I/O implementation *the IEEE Conference on Cluster Computing*
[11] Liao W, Ching A, Coloma K, Choudhary A and Ward L 2007 An Implementation and Evaluation of Client-Side File Caching for MPI-IO *the International Parallel and Distributed Processing Symposium*
[12] Ross R, Latham R, Gropp W, Thakur R and Toonen B 2005 Implementing MPI-IO Atomic Mode Without File System Support *the 5th IEEE/ACM International Symposium on Cluster Computing and the Grid*
[13] IEEE/ANSI Std 10031 1996 *Portable Operating System Interface (POSIX)-Part 1: System Application Program Interface (API) [C Language]*
[14] Yu H, Sahoo R, Howson C, Almasi G, Castanos J, Gupta M, Moreira J, Parker J, Engelsiepen T, Ross R, Thakur R, Latham R and Gropp W D 2006 High Performance File I/O for the BlueGene/L Supercomputer *the 12th International Symposium on High-Performance Computer Architecture (HPCA-12)*
[15] Almasi G, Archer C, Castanos J, Erway C, Heidelberger P, Martorell X, Moreira J, Pinnow K, Ratterman J, Smeds N, Steinmacher-burow B, Gropp W and Toonen B 2004 Implementing MPI on the BlueGene/L Supercomputer *Euro-Par Conference on Parallel Processing*
[16] Loft R D 2005 Blue Gene/L Experiences at NCAR *IBM System Scientific User Group meeting (SCICOMP11)*
[17] Nisar A, Liao W and Choudhary A 2008 Scaling Parallel I/O Performance Through I/O Delegate and Caching System *International Conference for High Performance Computing, Networking, Storage and Analysis*
[18] Thakur R, Gropp W and Lusk E 1997 *Users Guide for ROMIO: A High-Performance, Portable MPI-IO Implementation* Technical Report ANL/MCS-TM-234, Mathematics and Computer Science Division, Argonne National Laboratory
[19] HDF Group *Hierarchical Data Format, Version 5* The National Center for Supercomputing Applications http://hdf.ncsa.uiuc.edu/HDF5
[20] del Rosario J, Brodawekar R and Choudhary A 1993 Improved Parallel I/O via a Two-Phase Run-time Access Strategy *the Workshop on I/O in Parallel Computer Systems at IPPS '93* pp 56–70
[21] Kotz D 1997 *ACM Transactions on Computer Systems* **15** 41–74
[22] Liao W and Choudhary A 2008 Dynamically Adapting File Domain Partitioning Methods for Collective I/O Based on Underlying Parallel File System Locking Protocols *International Conference for High Performance Computing, Networking, Storage and Analysis*
[23] Hertel E, Bell R, Elrick M, Farnsworth A, Kerley G, Mcglaun J, Petney S, Silling S, Taylor P and Yarrington L 1993 CTH: A Software Family for Multi-Dimensional Shock Physics Analysis *the 19th International Symposium on Shock Waves* pp 377–382
[24] McGlaun J, Thompson S and Elrick M 1989 CTH: A three-dimensional shock wave physics code *the Hypervelocity Impact Symposium* pp 12–14
[25] Klundt R, Weston M and Ward L 2008 I/O Tracing on Catamount Tech. Rep. SAND2008-3684 Sandia National Laboratories Albuquerque, New Mexico
[26] Riedel E, Gibson G and Faloutsos C 1998 Active storage for large-scale data mining and multimedia *International Conference on Very Large Data Bases* (San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.) pp 62–73 ISBN 1-55860-566-5
[27] Keeton K, Patterson D and Hellerstein J 1998 *SIGMOD Record* **27** 42–52
[28] Acharya A, Uysal M and Saltz J 1998 Active disks: programming model, algorithms and evaluation *International Conference on Architectural support for programming languages and operating systems* (New York, NY, USA: ACM) pp 81–91 ISBN 1-58113-107-0
[29] Lim H, Kapoor V, Wighe C and Du D 2001 *Mass Storage Systems, IEEE Symposium on* **0** 101 ISSN 1051-9173
[30] Felix E, Fox K, Regimbal K and Nieplocha J 2006 Active storage processing in a parallel file system *International Conference on Linux Clusters: The HPC Revolution*