Achievements and challenges for I/O in computational science

Robert Ross[1], **Rajeev Thakur**[1], **and Alok Choudhary**[2]

[1]Mathematics and Computer Science Division, Argonne National Laboratory, Argonne IL 60439

[2]Electrical and Computer Engineering Department,Northwestern University, Evanston IL 60208

E-mail: `rross@mcs.anl.gov, thakur@mcs.anl.gov, choudhar@ece.northwestern.edu`

Abstract. The data access needs of computational science applications continue to grow, both in terms of the rates of I/O necessary to match compute capabilities, and in terms of the features required of I/O systems. Particularly, wide-area access to data, and moving data between systems, has become a priority.

Key achievements in I/O for computational science have enabled many applications to effectively use I/O resources. However, growing application requirements challenge I/O system developers to create solutions that will make I/O systems easier to use, improve performance, and increase the manageability. In this work we outline the recent achievements and current status of I/O systems for computational science. We then enumerate key challenges for I/O systems in the near future and discuss ongoing efforts that address these challenges.

1. Introduction

Computational science applications rely increasingly on I/O subsystems. Some applications have I/O-intensive initialization phases where large input datasets are read prior to execution. These input datasets vary widely in size and format. Applications often save their state (checkpoint) during execution so that some progress is saved in the event that a system failure occurs during the run. This state could be Gbytes to Tbytes in size, and often it is never accessed again. For visualization purposes, applications often save key values at regular periods during execution. The amount of data stored for each period is often smaller than a checkpoint, but this data is often recorded more frequently. This data is often read many more times than it is written as scientists observe the results of their work. Application scientists often perform post-processing or visualization on a different system than the one on which simulations were performed. This might allow the scientists to save allocated time on the machine for future simulations, or it might facilitate interactive visualization that is not possible using remote resources. Increasingly wide-area access to data is becoming a necessary aspect of computational science workflows.

While the capacity of I/O systems has been increasing at a rate commensurate with increases in processing power and network bandwidths, the same cannot be said for the bandwidth of I/O systems. Because of this phenomenon, the most obvious area for improvement in I/O systems has been in the area of performance. Parallel I/O systems have been developed to address the problem of performance. Parallel I/O systems combine many individual components (e.g.

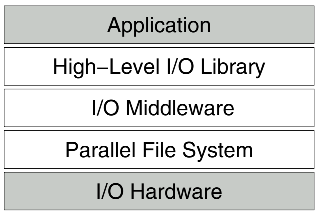

Figure 1. I/O software stacks provide the connection between applications and I/O hardware.

disks, servers, network links) together into a coherent whole used to provide high aggregate I/O performance to parallel applications.

While it is easy for us to concentrate on the performance of I/O systems for high-performance computing (HPC) as the key issue in I/O system development, there are other issues that are equally important. Another key issue is usability: how easy it is for computational science applications to leverage I/O resources. This determines how likely applications are to be able to take advantage of the resources. Finally, increasing the reliability and maintainability of these systems is critical. Without systems that are very reliable and easy to maintain, we run the risk of creating I/O systems that cannot be widely deployed. Fundamentally the mandate for I/O system developers is to provide reliable, easy-to-use, high-performance, and scalable I/O solutions for computational science.

There have been a number of recent, significant advances in I/O for computational science, but many challenges still remain. In this work we will outline both the achievements and challenges for I/O developers designing solutions for computational science. In Section 2 we will outline key accomplishments in the development of these I/O solutions. In Section 3 we will discuss issues that are critical to the future success of I/O systems in computational science. In Section 4 we will outline steps towards better, more usable I/O systems and point towards encouraging long-term possibilities.

2. I/O Achievements

Many improvements have been made in I/O systems both by the HPC community and by outside groups. One key category of improvements has been in the organization of I/O software and in defining standard interfaces to various layers, both software and hardware. Another important improvement has been the development and deployment of multiple parallel file system (PFS) implementations. Finally, many options for wide-area data movement have been developed.

2.1. Organizing I/O Software

I/O software has moved from monolithic serial I/O libraries to software stacks with at least three layers: parallel file system, I/O middleware, and high-level I/O interface. These three layers are shown in Figure 1. Each of these layers provides a subset of functionality; and in order to understand the role of the parallel file system in this layered system, we must first discuss the I/O stack as a whole.

The *high-level I/O library* library is responsible for applying structure to files in order to maintain a self-describing, portable data container and present a data abstraction to the application programmer that is close to the model used in the application. One example of such a library is Parallel netCDF [14]. Parallel netCDF (PnetCDF) is an extension to the serial netCDF library originally developed at Unidata [18]. It provides for the storage of multidimensional, typed datasets in a portable format, as well as storing attributes on this data. PnetCDF is built on the MPI-IO interface, part of the MPI-2 standard. The HDF5 [10] high-level I/O library is also widely used in computational science and also layers on top of MPI-IO.

The second key component of an I/O stack is *I/O middleware*. This component is responsible for providing the base on which high-level I/O libraries may be built. Typically this layer provides a mapping from the relatively simple interfaces of parallel file systems into an interface that leverages concepts from the programming model, such as communicators and datatype in the MPI programming model. The best example of I/O middleware is the MPI-IO interface [15], which is an achievement in itself discussed in Section 2.2.

The last component of this I/O stack is the *parallel file system*. Achievements in the area of parallel file systems are addressed in Section 2.4.

2.2. MPI-IO Interface

Parallel I/O performance has long been hampered by the lack of an appropriate, standard, portable interface for parallel I/O. In the absence of such an interface, users used the POSIX API, which is not well-suited for the I/O needs of parallel scientific applications. To meet this need, the MPI Forum defined an interface for parallel I/O as part of the MPI-2 Standard [15]. This interface is commonly referred to as MPI-IO. MPI-IO has many features specifically needed for parallel applications, such as, weaker consistency and atomicity semantics by default (although the user can optionally select strong semantics) and support for noncontiguous accesses and collective I/O. MPI-IO is widely available as all MPI implementations, both vendor-supported and freely available implementations, support it. It has become the default API for implementing high-level I/O libraries, such as HDF-5 and PnetCDF. Many applications also use MPI-IO directly.

2.3. Evolving Hardware Interfaces

We are beginning to see smarter I/O devices as well. Object-based storage devices are becoming standardized [2]. These devices aren't really "object-based storage" in the object-oriented sense; they do not provide strongly-typed data or allow for functions to be stored alongside data. However, they do allow named entities to be stored, so that data for a given "object" is understood to be associated with the object by the storage device. This interface leads to better organization at the device layer, and it simplifies the organization of tasks required of the parallel file system, allowing PFS developers to focus on higher-level functionality.

2.4. Parallel File Systems

The parallel file system (PFS) is responsible for managing all the storage hardware components. It presents a single logical view of this hardware that can be leveraged by other software layers. It also enforces a consistency model so that the results of concurrent access are well-defined. The PFS must scale to very large numbers of clients, handling many concurrent and seemingly independent operations. The PFS must present a rich interface on which efficient implementations of higher-level I/O components may be built. We expect these components to be the most frequent, and most important, users of any PFS interface.

Parallel file systems have existed for many years. However, only recently have parallel file systems emerged that will run on a variety of hardware platforms and operating systems. This change has resulted in active competition in the area of parallel file systems.

Three parallel file systems are currently being actively deployed and developed for HPC. The General Parallel File System (GPFS) from IBM grew out of the Tiger Shark multimedia file system [20] and has been widely used on the AIX platform. It is now also available on Linux clusters. The Lustre file system [5] is being developed by Cluster File Systems. It is in production use at a collection of Department of Energy laboratories and has been chosen for use on upcoming Cray systems. The PVFS2 parallel file system [19] is a second file system developed by the authors of the Parallel Virtual File System [6]. This system differs from the previous two in that it is an open, community-driven effort to build a parallel file system specifically for HPC and serves not only as a production PFS option but also as a vehicle for research in parallel I/O concepts. It is also available for Linux clusters.

Groups are also beginning to make direct comparisons of these systems [9]. As a result of the availability of multiple PFS options and the active comparison of these systems under identical conditions, it is now possible to quantitatively compare design and implementation options for parallel file systems.

2.5. Wide-Area Data Access

Computational science workflows often involve multiple sites. One issue in such workflows is data accessibility. Many tools have been created for moving data from one site to another.

GridFTP and the XIO interface are commonly deployed as part of the Globus Toolkit [1]. Storage Resource Managers allow access to remote data, particularly data existing on HPSS tape systems [21]. Logistical Networking software manages storage and retrieval of data from "depots" located on the internet [17]. The Storage Resource Broker also allows for remote access, but adds capabilities for organizing data into "collections" [4].

These systems are primarily designed for providing access to whole files, or sets of files, in the wide area. For this purpose they excel, attaining substantial fractions of available network bandwidth between sites. These tools have become integral parts of many application workflows.

3. I/O Challenges

While the aforementioned improvements have had a significant impact on the performance and usability of I/O systems for computational science, many challenges still remain that must be addressed. We describe some of these issues and discuss ongoing efforts that address these challenges.

3.1. Enhancing POSIX I/O

The POSIX I/O API [11] is the most widely used API for access to file systems, both by applications as well as by I/O libraries. The main reason for its widespread use is its widespread availability: It is available on all flavors of Unix as well as on Windows. Because it was the only option for many years, many parallel applications and libraries such as MPI-IO have been written to use the POSIX interface. The POSIX API, however, was originally designed for the needs of sequential applications and has many features that limit the performance of parallel applications.

For example, POSIX has a strong consistency and atomicity model, which states that a write from any process is visible to all other processes as soon as the write function returns. Furthermore, if concurrent writes from two processes overlap in the file, the result is the data written by either one process or the other, and nothing in between. These requirements constrain the implementation considerably: A process must acquire exclusive access to the range of bytes it accesses even if no other process may be concurrently accessing the same region (because it doesn't know that). Most parallel scientific applications do not write concurrently to overlapping regions of the file. If an optional weak consistency mode were available, such applications could select it for higher performance.

Many parallel applications need to access noncontiguous data regions in the file because of the difference between the data ordering in the file and data distribution among the processes of a parallel program. An example is distributed array access where the array is stored in the file in row-major order of the global array, and it is distributed among processes in a (block,block) manner. The POSIX `read` and `write` functions allow users to read and write only a single contiguous piece of data at a time. Writing noncontiguous data requires calling these functions multiple times, which is very expensive because of the high I/O latency. Instead, if POSIX had a `read_list` or `write_list` function that allowed the user to specify a noncontiguous access pattern with a single function, it would allow the implementation to optimize the noncontiguous request by using techniques such as data sieving [24]. We note that POSIX does have a function `lio_listio` that allows the user to specify a list of reads and writes, but each read or write is treated as a separate request, which prevents optimizations.

Many problems with using POSIX for parallel access have been identified by the community [7, 13], and proposals have been made for better building-block interfaces for I/O middleware [23]. An effort is underway by members of the I/O community from laboratories, universities, and industry to define extensions to the POSIX interface specifically for HPC. These extensions would provide more efficient mechanisms for accessing parallel file systems and would be a more effective interface for I/O middleware. Many of the concepts embodied in these extensions are being prototyped in the PVFS2 parallel file system [19, 13].

3.2. Application I/O Interfaces

The creation of high-level I/O interfaces was an important step in increasing usability for applications. Advances in algorithms and data models have resulted in significant performance gains for applications; however, I/O interfaces have not yet evolved to provide interfaces that match well with these new data models. As a result, application programmers must manually map their data structures into the ones that are supported by I/O interfaces (e.g. multidimensional datasets).

The next step is to create more domain-specific I/O interfaces (e.g. a climate modelling interface) or more data model-specific I/O interfaces (e.g. an adaptive mesh interface). These interfaces would further enhance the usability of I/O system software. These more specific interfaces would be built on top of the existing I/O stack, building on HDF5, PnetCDF, or some similar library. This allows for the fastest implementation and greatest possible reuse of existing infrastructure.

The real challenge in application I/O interfaces is in their tight coupling to applications or groups of applications. Neither application developers nor I/O system software developers have the knowledge and expertise to create these new interfaces alone. Active collaboration will be required to bridge this final interface gap between applications and I/O systems.

3.3. High-Level I/O Library Performance

A remaining issue with high-level I/O interfaces is that of performance. Second-generation high-level I/O interfaces, such as PnetCDF, have shown significant performance improvements over the first-generation interfaces (e.g. HDF5) by learning from experiences with these earlier systems [14]. Nevertheless, performance of even these newer libraries trails far behind the peak performance possible from the underlying storage system. As a result, applications must make tough decisions between usability and performance when choosing what interface to use to access storage.

A number of avenues exist through which the performance of high-level I/O libraries might be improved. One option is the inclusion of advanced caching systems into the high-level I/O library itself [8]. The high-level I/O library is an ideal layer at which to place such caching because a great deal of information about data layout is known at this layer and lost below. For example, only the high-level I/O library knows the relationship between the variables and structures stored in the file and the byte locations in the file. Thus, better decisions can be made about what to cache and what not to cache, leading to better overall performance [25].

Another option is the use of alternative file layouts. A major source of inefficiency in high-level I/O data formats is the perturbation of data layout due to metadata in the file itself. By better matching the distribution of metadata stored in the file to the layout of the file on the file system, we might achieve better overall performance. Further opportunities exist for removing metadata from the file byte stream entirely as extended attribute support in file systems becomes more common.

3.4. Data Discovery

There is a split in the community at this time related to I/O requirements: some applications choose to use databases for storage, while others use the I/O stacks and file systems discussed in this work. It appears to be the case that many applications could benefit from some database-like functionality in I/O systems, but it is not clear that full relational databases are necessary for most applications.

Query operations are becoming more common as data sizes grow. At this time, none of the I/O stack components we have described provide any query capabilities over user data. Different indexing techniques, such as bitmap indexing, are being investigated as methods for finding data items in files [26], but these cannot be implemented in a portable way without some mechanism for describing the semantics of file data.

High-level I/O libraries are the obvious place to implement query operations and optimizations, because they have knowledge of the semantics of the data in the file and when and how that data is being changed. With some work, even existing high-level I/O libraries could be augmented to provide both interfaces for query operations and capabilities for storing indices within the file itself, speeding up these operations. Data mining operations have similar requirements, and would likewise best be implemented at the high-level I/O layer.

At the same time, performing scanning operations at the storage device is very compelling because it could result in a significant reduction in the amount of data moved over the storage network. More research is necessary to better understand how we might implement such functionality, or support such functionality, in the parallel file system or in I/O hardware.

3.5. Wide-Area Data Access

As mentioned in Section 2.5, a number of systems for wide-area data movement have been created. However, in general no common programmatic interface is available for accessing these. What is needed is a way to access these data resources through our existing I/O stack. Three projects are in progress and working towards this goal.

The most obvious method for hooking wide-area data tools into the I/O stack is through new implementations of MPI-IO. The MPI-IO/SRM project is working to provide an MPI-IO interface to SRM storage resources, and the MPI-IO/LN project is working to create logistical network access through MPI-IO. Both of these efforts fall under the Scientific Data Management SciDAC. Implementations have also been prototyped for access to SRB resources through MPI-IO and GridFTP [3] through MPI-IO.

Because these implementations provide MPI-IO access, we can layer PnetCDF and HDF5 on top of them to provide wide-area access to these file formats. Performance will be an issue with these access methods. The majority of the wide-area access tools were designed for moving whole files, not the many small pieces we often see accessed during application I/O. More research is necessary to determine the appropriate optimizations for this combination of high-latency data access and application I/O patterns.

3.6. I/O System Management and Reliability

The sheer number of I/O devices used in today's I/O subsystems leads to many opportunities for failures to occur. I/O systems are now being used as resources for multiple clusters as well and thus are required to operate over multiple networks and in heterogeneous environments, complicating the implementation and configuration of such systems. Installation, configuration, and tuning of these systems verges on requiring an expert. To address these issues, more intelligence is needed in the I/O system, particularly in the parallel file system.

One way to integrate increased "intelligence" in a parallel file system is to augment servers so that they can communicate with one another. Once this is possible, servers can exchange information about their health and their resource utilization. Using this information the collection of servers could work towards specific goals, such as balancing data load or maximizing access performance for frequently accessed files. This concept of self-managing, self-tuning storage is what is known as *autonomic storage*, which is just one example of autonomic computing [12]. A system integrating these concepts is easier to manage because many tasks are performed by the system itself.

In parallel file systems, integrating autonomic concepts is made more complicated by the need for predictable performance. The additional communication and data movement inherent in autonomic storage could cause performance to vary widely even for identical access patterns. This type of behavior is unacceptable in the tightly scheduled HPC systems of today.

Finally, more work is necessary to address failure tolerance in HPC I/O systems. Techniques such as RAID [16] can be used to hide disk failures, and high-availability cluster can be used to provide fail-over in the event of server failures. However, client failures are still a significant issue, particularly so when clients cache data and metadata that is important to the I/O system (which they do for systems such as GPFS and Lustre). When clients fail, many systems must drop into a recovery mode to determine what was lost before continuing, perturbing performance for the system as a whole.

The NFS [22] protocol, used for network access to single I/O servers, uses a *stateless* client system. With this approach client failures do not cause the loss of important file system data and may be ignored. The PVFS2 system takes a similar approach. Adoption of this approach, or further research into client caching, is necessary to reduce the impact of this last type of failure on the system as a whole.

4. Conclusions

Many significant improvements have been made in the area of HPC I/O in recent years that directly address the needs of computational science and position us to tackle remaining and future issues. In particular increased competition in the area of I/O is good for applications. We see competition now in two areas: parallel file systems and high-level I/O libraries, and performance and usability are increasing as a result of this competition.

Standards mean that applications can be more easily ported to new I/O systems, and they facilitate this competition. The extensions to POSIX I/O, the ANSI T-10 OSD proposal, and also the pNFS proposed extensions to the NFSv4 standard are all important steps in the direction of high-performance, standards-based I/O systems. The next step in increasing the usability of I/O systems for computational science is going to be through the development of domain or data model specific interfaces. Development of these interfaces must be a team effort: neither application scientists nor I/O system software developers have the comprehensive knowledge necessary to build effective solutions in this space.

Management and reliability of these systems, while not a critical issue at this time, will grow as an issue as the scale of these systems increases. Investigation of concepts like autonomics for the purpose of improving the production quality of our I/O systems.

In summary, it should be obvious that I/O systems and I/O system software are still evolving to meet the challenges of existing and future HPC systems. By scaling to new system sizes, integrating new interfaces to increase usability, spanning wider areas, and managing themselves, I/O systems can become more useful and less expensive components than ever before.

Acknowledgments

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract W-31-109-Eng-38.

References

[1] Bill Allcock, Joe Bester, John Bresnahan, Ann L. Chervenak, Ian Foster, Carl Kesselman, Sam Meder, Veronika Nefedova, Darcy Quesnel, and Steven Tuecke. Data management and transfer in high-performance computational grid environments. *Parallel Computing*, 28(5):749–771, May 2002.

[2] ANSI INCITS T10 Committee. Project t10/1355-d working draft: Information technology – SCSI object-based storage device commands (OSD), July 2004.

[3] Troy Baer and Pete Wyckoff. A parallel I/O mechanism for distributed systems. In *Proceedings of the 2004 IEEE International Conference on Cluster Computing*, pages 63–69, San Diego, CA, September 2004. IEEE Computer Society Press.

[4] Chaitanya Baru, Reagan Moore, Arcot Rajasekar, and Michael Wan. The SDSC storage resource broker. In *Proceedings of the 1998 conference of the Centre for Advanced Studies on Collaborative research*, 1998.

[5] Peter J. Braam. The lustre storage architecture. Technical report, Cluster File Systems, Inc., 2003.

[6] Philip H. Carns, Walter B. Ligon III, Robert B. Ross, and Rajeev Thakur. PVFS: A parallel file system for linux clusters. In *Proceedings of the 4th Annual Linux Showcase and Conference*, pages 317–327, Atlanta, GA, October 2000. USENIX Association.

[7] Avery Ching, Alok Choudhary, Wei keng Liao, Robert Ross, and William Gropp. Noncontiguous i/o through pvfs. In *Proceedings of the 2002 IEEE International Conference on Cluster Computing*, September 2002.

[8] Kenin Coloma, Alok Choudhary, Wei keng Liao, Lee Ward, and Sonja Tideman. DAChe: Direct access cache system for parallel I/O. In *to appear in Proceedings of the 2005 International Supercomputer Conference*, Heidelberg, Germany, June 2005.

[9] J. Cope, M. Oberg, H.M. Tufo, and M. Woitaszek. Shared parallel file systems in heterogeneous linux multi-cluster environments. In *Proceedings of the 6th LCI International Conference on Linux Clusters*, April 2005.

[10] HDF5 wins 2002 R&D 100 award. `http://hdf.ncsa.uiuc.edu/HDF5/RD100-2002/All_About_HDF5.pdf`.

[11] IEEE/ANSI Std. 1003.1. Portable operating system interface (POSIX)–part 1: System application program interface (API) [C language], 1996 edition.

[12] Jeffrey Kephart and David Chess. The vision of autonomic computing. *IEEE Computer*, pages 41–50, January 2003.

[13] Robert Latham, Robert Ross, and Rajeev Thakur. The impact of file systems on MPI-IO scalability. In *Proceedings of EuroPVM/MPI 2004*, September 2004.

[14] J. Li, W.-K. Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, and R. Latham. Parallel netCDF: A scientific high-performance I/O interface. Technical Report ANL/MCS-P1048-0503, Mathematics and Computer Science Division, Argonne National Laboratory, May 2003.

[15] Message Passing Interface Forum. MPI-2: Extensions to the message-passing interface, July 1997. `http://www.mpi-forum.org/docs/docs.html`.

[16] David Patterson, Garth Gibson, and Randy Katz. A case for redundant arrays of inexpensive disks (RAID). In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 109–116, Chicago, IL, June 1988. ACM Press.

[17] James S. Plank, Terry Moore, and Micah Beck. An end-to-end approach to globally scalable network storage. In *Proceedings of ACM SIGCOMM 2002*, Pittsburgh, PA, August 2002.

[18] Russ Rew and Glenn Davis. Data management: NetCDF: an interface for scientific data access. *IEEE Comput. Graph. Appl.*, 10(4):76–82, 1990.

[19] Robert Ross, Robert Latham, Neill Miller, Philip Carns, Walter B. Ligon III, Pete Wyckoff, and Troy Baer. The PVFS2 parallel file system. In *Submitted to Cluster Computing 2005*, September 2005.

[20] Frank Schmuck and Roger Haskin. GPFS: A shared-disk file system for large computing clusters. In *First USENIX Conference on File and Storage Technologies (FAST'02)*, Monterey, CA, January 28-30 2002.

[21] A. Shoshani, A. Sim, and J. Gu. Storage resource managers: Middleware components for grid storage. In *Proceedings of the Nineteenth IEEE Symposium on Mass Storage Systems (MSS '02)*, 2002.

[22] Hal Stern. *Managing NFS and NIS*. O'Reilly & Associates, Inc., 1991.

[23] Rajeev Thakur, William Gropp, and Ewing Lusk. On implementing MPI-IO portably and with high performance. In *Proceedings of the 6th Workshop on I/O in Parallel and Distributed Systems*, pages 23–32. ACM Press, May 1999.

[24] Rajeev Thakur, William Gropp, and Ewing Lusk. Optimizing noncontiguous accesses in MPI-IO. *Parallel Computing*, 28(1):83–105, January 2002.

[25] Murali Vilayannur, Anand Sivasubramaniam, Mahmut Kandemir, Rajeev Thakur, and Robert Ross. Discretionary caching for I/O on clusters. In *Proceedings of the Third IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 96–103, Tokyo, Japan, May 2003. IEEE Computer Society Press.

[26] Kesheng Wu, Wendy Koegler, Jacqueline Chen, and Arie Shoshani. Using bitmap index for interactive exploration of large datasets. In *In Proceedings of SSDBM 2003*, 2003.