Input/Output APIs and Data Organization for High Performance Scientific Computing

Jay Lofstead
College of Computing
Georgia Institute of Technology
Atlanta, Georgia
lofstead@cc.gatech.edu

Fang Zheng
College of Computing
Georgia Institute of Technology
Atlanta, Georgia
fzheng8@mail.gatech.edu

Scott Klasky
Oak Ridge National Laboratory
Oak Ridge, Tennessee
klasky@ornl.gov

Karsten Schwan
College of Computing
Georgia Institute of Technology
Atlanta, Georgia
schwan@cc.gatech.edu

***Abstract*—Scientific Data Management has become essential to the productivity of scientists using ever larger machines and running applications that produce ever more data. There are several specific issues when running on petascale (and beyond) machines. One is the need for massively parallel data output, which in part, depends on the data formats and semantics being used. Here, the inhibition of parallelism by file system notions of strict and immediate consistency can be addressed with 'delayed data consistency' methods. Such methods can also be used to remove the runtime coordination steps required for immediate consistency from machine resources like Bluegene's separate networks for barrier calls and its dedicated IO nodes, thereby freeing them to instead, perform alternate tasks that enhance data output performance and/or richness. Second, once data is generated, it is important to be able to efficiently access it, which implies the need for rapid data characterization and indexing. This can be achieved by adding small amounts of metadata to the output process, thereby permitting scientists to quickly make informed decisions about which files to process from large-scale science runs. Third, failure probabilities increase with an increasing number of nodes, which suggests the need for organizing output data to be resilient to failures in which the output from a single or from a small number of nodes is lost or corrupted.**

This paper demonstrates the utility of using delayed consistency methods for the process of data output from the compute nodes of petascale machines. It also demonstrates the advantages derived from resilient data organization coupled with lightweight methods for data indexing. An implementation of these techniques is realized in ADIOS, the Adaptable IO System, and its BP intermediate file format. The implementation is designed to be compatible with existing, well-known file formats like HDF-5 and NetCDF, thereby permitting end users to exploit the rich tool chains for these formats. Initial performance evaluations of the approach exhibit substantial performance advantages over using native parallel HDF-5 in the Chimera supernova code.

I. Introduction

Scientific data formats like HDF-5 and NetCDF have given rise to rich tool chains for use by science end users. Performance issues with their direct use by petascale (and beyond) applications, however, demonstrate the need for (1) improvements in scalability with respect to the degree of attainable parallelism, a specific technique used in this paper being delayed data consistency, (2) the ability to perform rapid data characterization to improve scientific productivity, and (3) resilience to isolated failures through improved data organization.

Delayed consistency is well-known to improve the performance of file systems. This paper explores its advantages outside such regular IO paths, by applying it to the process of producing output data on the compute nodes of petascale machines. Here, by delaying consistency computations, the total time taken by each compute node to complete its IO is no longer affected by the completion time of other nodes and/or by the control operations like the broadcast calls performed by the current implementation of HDF-5. Eliminating control operations also obviates the use of precious machine resources like Bluegene's separate control network for purposes like these, and it removes associated computations and delays from IO nodes. For instance, the current implementation of parallel HDF-5 uses these resources for broadcast calls to ensure data consistency. Finally, the use of and need for such specialized hardware introduces issues with code portability. An example is the exposure of its coordination infrastructure to science codes by the MPI ADIO layer. By instead, hiding these behind a simple higher level API, alternative coordination mechanisms and collective operations (e.g., delayed consistency methods) can be implemented without necessitating code changes.

While potentially improving application performance, techniques like delayed consistency do not directly contribute to the productivity of the scientists running these codes. Here, it is useful to associate additional functionality with data output, such as lightweight, compact mechanisms for data characterization and indexing. This has been recognized by others, resulting in index methods for HDF-5 files, like FastBit [1] and PyTables [2], which each provide various levels of detail about data to aid in rapid access. Unfortunately, these methods have not been added to the format and API specification. As a

result, we advocate and evaluate an alternative approach in which it is possible to compute and associate with output simple data characteristics, an example being max and min values for an array being output, so that scientists can rapidly decide upon the value of further or immediately processing this data. Further, to control or eliminate the additional overhead potentially introduced by the associated computations, there is flexibility in 'where' and 'when' data characteristics are computed: immediately associated with output, in a data staging area (i.e., on compute nodes), in an IO Graph [3] as part of additional processing as data flows to disks or other destinations, and finally, by metabots [4] when data is already stored on disk. Such operations could even be performed in a traditional workflow system (e.g., Kepler [5], Pegasus [6], or DagMAN [7]) as part of more complex data processing actions. Our current implementation performs data characterization on compute nodes, with minimal impact on IO performance. The data characterizations (e.g., statistical measures) are stored in the same file as the actual output data, but end users can also place them into secondary files, in order to avoid having to load and scan, say, multiple tapes on a tape system like HPSS [8] to assess the value of data to a certain computational or analysis action.

With current and next generation high performance machines, the probability is high that one of many nodes performing data output fails to complete that action. While the loss of that node's data may be acceptable to the scientific application, the failure of all nodes to complete their output due to a single node's problems is not. Instead, data output should be implemented to be robust to failures, adopting principles from other areas of Computer Science (e.g., consider the Google file system [9]). In response to these needs, we have implemented the BP file format as part of ADIOS, the adaptable IO system [10]. Analogous to proxy processes in other systems, the idea of the BP 'proxy' data format is to act as an *intermediate data format* used during data output and initial processing (e.g., the aforementioned lightweight data characterization). We call BP an intermediate format because typically, converters will be applied to BP files so that subsequent data analysis and storage can leverage the extensive tool chains existing for popular formats like HDF-5 and NetCDF. This paper evaluates the performance of an HDF-5 converter as part of an output operation.

To summarize, this paper makes the following contributions:

1) use of delayed consistency and other techniques for enhanced parallelism, supported by a high level API for IO;
2) lightweight methods for data characterization integrated into the process of data output;
3) introduction of an 'intermediate file format' to store raw data and its characterizations in ways resilient to node failures; and
4) compatibility with common file formats like HDF-5 and NetCDF.

The rest of the paper is organized as follows. Section II describes related work. Section III contains an architectural description. Section IV has performance results. Conclusion and future work appear in Section V, followed by references.

II. Related Work

Related work exists in three areas: relaxed consistency systems, data indexing, and IO APIs.

Delayed consistency has previously been studied for file systems. For example, the Lightweight File Systems [11] project at Sandia Labs has stripped down POSIX semantics to a core of authentication and authorization affording layering of other semantics, like consistency, on an as-needed basis. Other file systems like the Serverless File System [12] have distributed metadata weakening the immediate consistency across the entire network of machines. NFS [13] relies on write-back local caches limiting the globally consistent view of the file system to the last synchronization operation. Our work considers the use of delayed consistency within single, large-scale files, the intent being to 'fix them up' whenever possible without inhibiting the performance of the petascale application.

Efficient techniques for indexing HDF-5 files are offered by FastBit [1], which uses a bitmap to indicate the presence of different values or value ranges within a given data element. PyTables [2] extends this concept to use a traditional relational database as a full content index. Unfortunately, these approaches have not been integrated into the base HDF-5 system, perhaps because of potential performance penalties when adding rich indices to large-scale files. Such penalties are our principal motivation for advocating simple, lightweight data characterization rather than full indexing methods in the BP intermediate file format.

While offering rich tool chains, there have always been scalability challenges for the NetCDF and HDF APIs and file formats. For terascale machines, such challenges were successfully addressed by moving from serial to parallel APIs. In fact, in many cases, the performance of parallel HDF-5 or NetCDF API has been exemplary [14]. However, additional options will have to be explored for petascale machines (and beyond), in part because of the high costs of the collective IO operations required for consistency enforcement. Such enforcement is necessary for providing a single coherent view of the globally distributed file data. We believe that the delayed consistency approach can be used to incur these costs so as to not inhibit parallel program performance.

As with our IO system, ADIOS, the ADIO layer of MPI provides some facility for hiding the underlying hardware and software environment from application codes, to provide portability and insulation from these details. However, since the MPI-IO API has explicit semantics exposed to the scientific code, certain operations, like collective writes, cannot be changed without impacting the host code. Our approach takes the further step of abstracting away details like coordination for collective operations thereby giving the implementation layer the flexibility to employ whatever techniques and features are available on a given platform without impacting host

sources.

III. Software Architecture

A. Architecture of ADIOS Layer

The ADIOS layer show in Figure 1 has several important characteristics that help us attain portable and scalable implementations of delayed consistency, lightweight data characterization, and output resilience. One such characteristic is that ADIOS hides all consistency-related functions of lower level APIs by moving them into metadata specifications located in an external XML file. The file selects for each IO group, the IO method to be used with the intent being to flexibly choose IO methods to best match the IO patterns of certain code output actions (e.g., restart files vs. intermediate results) and local machine performance characteristics. IO methods also include the aforementioned lightweight data characterization, where currently, the only method offered is one that immediately upon data output computes simple characteristics like mix/max array values. Future work will examine alternative methods that place such computations at different 'locations' in or outside the IO path. Finally, the BP intermediate file format's current implementation is designed to maximize parallel output performance while also containing sufficient information to (later) validate data consistency.

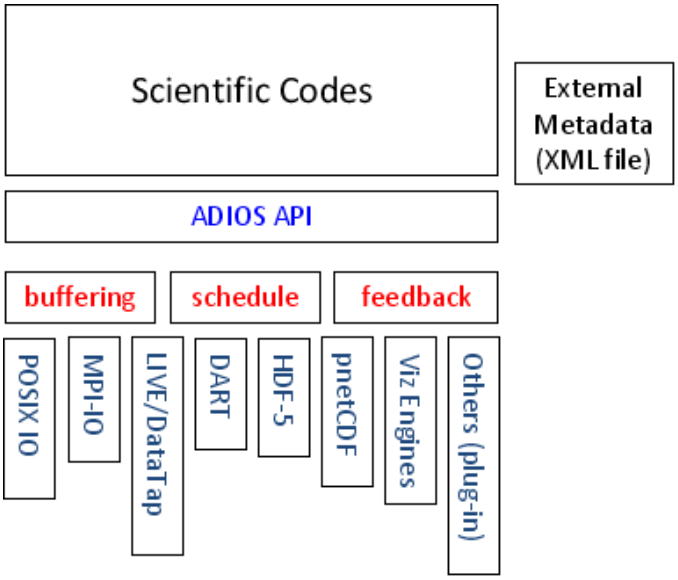

Fig. 1. ADIOS Architecture

Our delayed internal file consistency implies synchronization only at the start and end of each IO operation. As a result, per node variations in process performance and/or in messaging delays experienced by consistency operations are avoided. The actual delay incurred corresponds to the longest total IO time for any single process rather than the sum of the longest times for any process for each step in the entire IO action. Further, by not exposing the actual calls made to the science code, end users can freely choose between using a fully consistent parallel API (e.g., during code development) vs. our delayed consistency methods (e.g., during high end production runs). Similarly, users can first employ MPI-IO or POSIX IO and then convert from the BP format to either HDF-5 or NetCDF, as needed. All such choices simply entail changing one entry in the ADIOS XML configuration file.

Our principal motivation for associating lightweight data characterization with output actions is to improve scientists' productivity, which is being reduced by the need to move across wide area links the ever larger datasets produced by petascale codes. Data characterization affords them with the opportunity to quickly inspect output data, to assess whether such movements are important before performing them. We term such actions data characterization because they can be user-specified and need not be complete, thereby avoiding the known high overheads of data indexing like that used by FastBit and PyTables. Further, entries for data characterization are made default parts of the BP intermediate file format, much like 'attributes' in other systems [15], so that end users need not be concerned with format changes or updates when data characterizations are changed.

As our primary research examples have all been write intensive science codes, we have focused our initial work at delivering the best possible write performance while still affording good read performance. We collect sufficient information during the write operations to enable good performing read operations, but we have not spent any time optimizing this portion of the code. Additional techniques for scalable, high performance reading are likely necessary and partially discussed in the future work section.

B. BP Architecture

The BP file format was specifically designed to support delayed consistency, lightweight data characterization, and resilience. The basic file layout is shown in Figure 2.

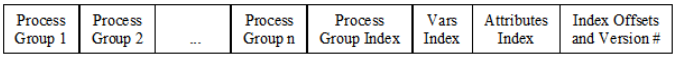

Fig. 2. BP File Layout

Each process writes its own output into a *process group* slot. These slots are variably sized based on the amount of data required by each process. Included in each process output are the data characteristics for the variables. For performance, we are investigating the advantages of padding these slots to file system whole stripe sizes. These and other size adjustments are possible and centrally managed during the file open process. This flexibility will be required to get the best possible performance from an underlying transport and file system.

The three index sections are stored at the end for ease of expansion during append operations. Their manipulation is currently managed by the root process of the group performing IO. The overhead of these indices is acceptably small even for a large number of processes. For example, for 100,000 processes and a large number variables and attributes in all process groups, such as 1000, the total index size will be on the order of 10 MB. Given the total size of the data from an

output operation of this size, 10 MB constitutes little more than a rounding error. Since these are at the end of the file, we reserve the last 28 bytes of the file for offset locations and for version and endian-ness flags.

Delayed consistency is achieved by having each process write independently with sufficient information to later validate that the consistency was not violated. While the replication of this data may not seem desirable, consider the ramifications of a three orders of magnitude performance penalty for instead, maintaining a single copy or consider the potential that the single copy being corrupted renders the entire output useless. We have measured the overhead per process to be on the order of a few hundred bytes for a few dozen variables. This cost, we believe, is well worth the time savings and greater resilience to failure.

Data characteristics are replicated into the indices stored at the end of the file. As mentioned above, the location of the index is stored at a known offset from the end of the file, thereby making it easy to seek to the index. Since the index is internally complete and consistent, it can be separated out and queried to determine if the associated data contains the desired features.

The BP format addresses resilience in two ways. First, once the initial coordination to determine file offsets is complete, each process can output its entire data independently and close the local connection to the file. This will commit the local data to storage, which constitutes some level of safety. Afterwards, ADIOS gathers all of the index data for each single output to the root process of the output operation, merges it together, and writes it to the end of the file as a footer. This merging operation is strictly appending various linked lists together making it efficient to perform. Second, the replicated metadata from each process in the footer gives a list of offsets to where each process group was written. Should this fail, it is possible to linearly search the file and determine where each process group begins and ends.

IV. Experimental Evaluation

Technical evaluations demonstrate the following. First, we discuss our practical experiences with the Chimera supernova code and its IO. We also identify the reasons for the existence of orders of magnitude differences in performance when using alternative methods for IO. Second, we demonstrate the lightweight nature of data characterization, by comparing the times taken to collect base data characteristics against those experienced by external indexing schemes and by a full data scan from a local disk. Finally, we discuss how the BP file format and the ADIOS API jointly achieve resilience to failures.

Evaluations are performed on two different machines: (1) Jaguar, the Cray XT4 system at Oak Ridge National Laboratory, and (2) Ewok, the Infiniband and Linux based end-to-end cluster at ORNL.

A. Delayed Consistency and Abstracted API

We evaluate the Chimera supernova code with weak scaling using both the native HDF-5 calls used in that code vs. our ADIOS calls for POSIX, for independent MPI-IO, and for collective MPI-IO. Results appear in Figure 3. The two most important items to notice in the graph are (1) that the parallel HDF-5 calls no longer scale linearly beyond 2,048 processes and (2) that the performance difference for 16K processes is three orders of magnitudes when compared with POSIX IO. We note that these runs use parallel HDF-5 tuned to use independent MPI-IO as the transport and all recommended performance options are enabled, based on the recommendations by IO experts at ORNL.

TABLE I
Parallel HDF-5

Parallel HDF-5		
Function	*# of calls*	*Total Time (sec)*
write	144065	33109.67
MPI_Bcast	314800	12259.30
MPI_File_open	2560	325.17
H5P, H5D, etc.	–	8.71
other	–	60

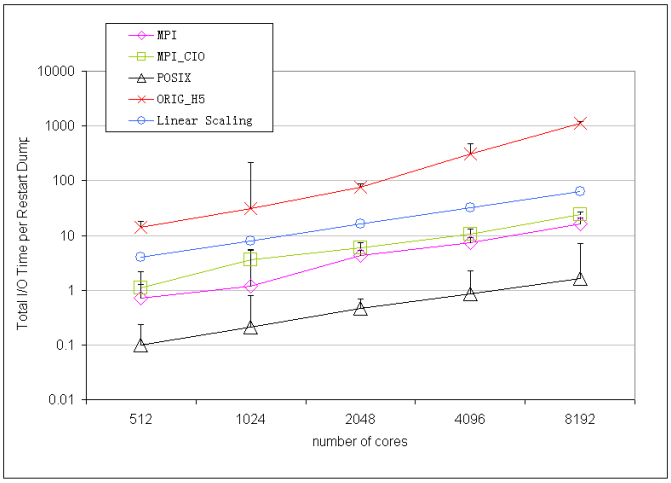

Fig. 3. Chimera Weak Scaling

A detailed profile of the HDF-5 vs. ADIOS calls is attained for a 512 process run. The tables below are the total time spent in all processes on each operation. Three key differences are apparent in Tables I and II. First, ADIOS does a cascaded MPI_Open call. The real cost of the open operation is the sum of the MPI_File_open and the MPI_File_recv. Even combined, they are less than the parallel open performed by HDF-5. The second and more striking is the number of writes and the time spent performing them. The third and most important for our delayed consistency argument is the MPI_Bcast calls to enforce consistency. ADIOS performance is improved by its ability to alter how the open operation is performed, buffering the write commands to perform larger, less frequent calls, and not requiring the broadcasts for consistency checking.

Briefly, the conversion time for a BP file generated from an 8192 process run into an HDF-5 format is only 117 seconds.

A more complete discussion of these results will appear in the final version of this paper. The final version of this paper

TABLE II
ADIOS INDEPENDENT MPI-IO

ADIOS Independent MPI-IO		
Function	*# of calls*	*Total Time (sec)*
write	2560	2218.28
MPI_File_open	2560	95.80
MPI_Recv	2555	24.68
other	–	65

will also describe (1) the performance impact of collecting the data characteristics during the IO process and compare that against both the worst case file scan and one or more of the various indexing prototypes available for HDF-5, and

(2) experiments that simulate different kinds of failures and their resulting impacts on output performance.

V. CONCLUSIONS AND FUTURE WORK

This paper demonstrates the utility of three basic ideas. First, it shows that the use of delayed consistency methods applied to the internals of large-scale files can result in up to three orders of magnitude performance improvements in IO on petascale machines. Second, by providing 'nearly free' data characterization as part of the base API, common questions like when a value or array reaches some threshold value can be answered without analyzing all of the output data. This aids both (1) in selecting which data to analyze from potentially slow storage like a tape library and also (2) in preserving the use of expensive analysis resources for the data most relevant to the scientific question at hand. Finally, resilience is achieved by replicating metadata to all process outputs. By using a footer index with replicated data from the process groups rather than a header, append operations are facilitated and we avoid relying on centralized metadata for file correctness.

REFERENCES

[1] L. Gosink, J. Shalf, K. Stockinger, K. Wu, and W. Bethel, "Hdf5-fastquery: Accelerating complex queries on hdf datasets using fast bitmap indices," in *In SSDBM*, 2006, pp. 149–158.

[2] H.-. P. Tables, "http://www.carabos.com/docs/opsi-indexes.pdf."

[3] P. M. Widener, M. Wolf, H. Abbasi, M. Barrick, J. Lofstead, J. Pullikottil, G. Eisenhauer, A. Gavrilovska, S. Klasky, R. Oldfield, P. G. Bridges, A. B. Maccabe, and K. Schwan, "Structured streams: Data services for petascale science environments," University of New Mexico, Albuquerque, NM, Tech. Rep. TR-CS-2007-17, November 2007. [Online]. Available: http://www.cs.unm.edu/ treport/tr/07-11/PetaScale.pdf

[4] P. M. Widener, M. Barrick, J. Pullikottil, P. G. Bridges, and A. B. Maccabe, "Metabots: A framework for out-of-band processing in large-scale data flows," in *Proc. 2007 International Conference on Grid Computing (Grid 2007)*, Austin, Texas, September 2007.

[5] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, "Scientific workflow management and the kepler system: Research articles," *Concurr. Comput. : Pract. Exper.*, vol. 18, no. 10, pp. 1039–1065, 2006.

[6] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," *Sci. Program.*, vol. 13, no. 3, pp. 219–237, 2005.

[7] G. Malewicz, I. Foster, A. Rosenberg, and M. Wilde, "A tool for prioritizing dagman jobs and its evaluation," *High Performance Distributed Computing, 2006 15th IEEE International Symposium on*, pp. 156–168, 0-0 2006.

[8] R. W. Watson and R. A. Coyne, "The parallel i/o architecture of the high-performance storage system (hpss)," in *MSS '95: Proceedings of the 14th IEEE Symposium on Mass Storage Systems*. Washington, DC, USA: IEEE Computer Society, 1995, p. 27.

[9] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 29–43, 2003.

[10] J. Lofstead, S. Klasky, K. Schwan, N. Podhorszki, and C. Jin, "Flexible io and integration for scientific codes through the adaptable io system (adios)," in *CLADE 2008 at HPDC*. Boston, Massachusetts: ACM, June 2008.

[11] R. Oldfield, L. Ward, R. Riesen, A. Maccabe, P. Widener, and T. Kordenbrock, "Lightweight i/o for scientific applications," *Cluster Computing, 2006 IEEE International Conference on*, pp. 1–11, 25-28 Sept. 2006.

[12] T. E. Anderson, M. D. Dahlin, J. M. Neefe, D. A. Patterson, D. S. Roselli, and R. Y. Wang, "Serverless network file systems," *ACM Trans. Comput. Syst.*, vol. 14, no. 1, pp. 41–79, 1996.

[13] NFS, "http://www.ietf.org/rfc/rfc3010.txt."

[14] W. Yu, J. Vetter, and H. Oral, "Performance characterization and optimization of parallel i/o on the cray xt," *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pp. 1–11, April 2008.

[15] F. Bustamante, G. Eisenhauer, K. Schwan, and P. Widener, "Efficient wire formats for high performance computing," in *In Proceedings of Supercomputing 2000*, 2000.

VI. ACKNOWLEDGEMENTS

This work was funded in part by Sandia National Laboratories under contract DE-AC04-94AL85000, a grant from NSF as part of the HECURA program, a grant from the Department of Defense, a grant from the Office of Science through the SciDAC program, and the SDM center in the OSCR office.