SIRIUS: A Revolutionary Approach to I/O and Storage for Exascale Scientific Computing

Your N. Here
Your Institution

Second Name
Second Institution

Abstract

The longest existing parallel I/O APIs, HDF5, PnetCDF, and NetCDF, are focused on treating the entire output from a paralell application as a single unit. These libraries work to gather the data from all processes and arrange it such that it is organized much like as if it were written from a single process. This approach, in particular the two-phase collective I/O with data sieving portion, has proven difficult to scale for 3-D domain decompositions at scale. ADIOS shifted the focus from treating the entire application output as a single entity to treating the output from each writer as a self-contained entity. This eliminated the need for the problematic two-phase collective I/O and has demonstrated good scalability well into the petascale era. Another shift is required to address the needs of exascale systems. New system I/O bandwidth is not keeping pace with the compute acceleration leading to accute bottlenecks. While "burst buffer" technology will help address the performance gap, it does not address the problem completely. This paper describes a new approach to not just an I/O library, but also the underlying storage infrastructure to address the needs of exascale applications cognizant of projected exascale platform characteristics and in the context of current industry trends.

1 Introduction

High performance computing simulations have driven tightly coupled platform scaling for decades. The output from these simulations are analyzed using regular tools and techniques optimized for this environment. The first generation of standardized APIs for scientific computing IO focused on serial output from a single process to a distributed or local file system. These systems still exist in the form of serial HDF5 [4] and the end-of-life NetCDF 3.x [13]. The key feature of these APIs was the logical data model that eliminated any parallel data decomoposition artifacts from the actual data storage. This offered consistent performance and regular data access patterns.

The second generation is really an evolution of the first with HDF-5 adding parallel capabilities and NetCDF bifurcating into NetCDF4 [14] and PnetCDF [8] with PnetCDF almost completely maintaining the data model and NetCDF4 shifting to use HDF5 underneath. The other improvement is incorporating knowledge of an underlying parallel file system to support large files written in unision from many processes.

The third generation is a reaction to the second based on recognizing that maintaining the logical data model in physical form became inordinately expensive [9] and offered a performance penalty in most cases when running at scale. Libraries like ADIOS [10] shifted from writing complete variables from distributed processes as a single entity to treating all data from a single source to be treated as a unit. This eliminated the data reorganization at the cost of more complicated reading patterns. In spite of this extra complication, the performance is generally better [9].

SIRIUS represents the next generation. New platforms are introducting additional storage layers to accelerate IO operations and to compensate for plateauing storage array performance. At the same time, data sizes continue to grow. The fourth generation IO API will have to both address the IO API and middleware layer as well as the deeper, more complex storage hierarchy. SIRIUS considers the local portion of each globally distributed variable as a management entity with data placement decisions across the entire storage hierarhy rather than on a single tier. Decisions on what data to place where is driven by the data contents itself according to end-user specified priorities and system characteristics. For example, if temperature and pressure are nearly always retrieved together, organizing the data to keep just these two arrays together without any interleaving from other variables or scalars will accelerate later reading. If these variables could also be "processed" such that the important data

features are maintained in high resolution while less interesting regions are stored in highly compressed–even using lossy compression–forms, the amount of data written can be greatly increased. While accelerating writing is important. This approach also supports accelerating reading.

This paper presents an overview of the efforts to build such a fourth generation storage and IO system. A discussion of the end-user API, storage challenges, quality of service features, and metadata challenges are all discussed briefly.

The rest of the paper is organized as follows. A brief overview of related work is presented first in Section 2. Section 3 discusses the differentiating platform features SIRIUS will provide. Section 4 discusses the challenges and opportunities for deploying scalable storage infrastructure. Section 5 discusses the quality of service features proposed for this project. Next is an exploration of the metadata challenges required to support such a system in Section 6. An initial prototype demonstration of the functionality is presented in Section 7. The paper is concluded in Section 8 with a summary of the broad issues covered in the paper.

2 Related Work

Ceph [17] is a distributed object store and file system. It offers both a POSIX and object interface including features typically found in parallel file systems. Ceph's unique striping approach uses pseudo-random numbers with a known seed eliminating the need for the metadata service to track where each stripe in a parallel file is placed. PVFS [3] offers optimizations to reduce metadata server load, such as a single process opening a file and sharing the handle. It has been commercialized in recent years as OrangeFS. Lustre [2] has become the de facto standard on most major clusters offering scalable performance and fine-grained end-user and programmatic control over how data is placed in the storage system. GPFS [15] offers a hands-off approach for providing good performance for scaling parallel IO tasks and is used extensively by its owner, IBM. Panasas [11] seeks to offer a dynamically adaptive striping system that detects the need for additional stripes for performance and adjusts the file layout as necessary.

Other file systems, like GoogleFS [6] and HDFS [16], address distributed rather than parallel computing and cannot be compared directly. The primary difference between distributed and parallel file systems is the ability of the file system to store and retrieve data simultaneously from multiple clients, in parallel, and treat the resulting collection of pieces as a single object. Distributed file systems rely on a single client creating a file, but distributing the set of files across a wide array of storage devices. The other, popular distributed file system of note is NFS [12] that has been used for decades for enterprise file systems.

3 Platform Features

The initial thoughts about an end-use API is to try to work within established APIs to ease user migration. However, some of the key features, such as the data "refactor" functionality and the quality of service requirements for both writing and reading are new. To manage these new requirements, an extended API is proposed. At a high level, the common API broad categories are discussed.

Refactoring may be a uniform operation across an entire variable. For example, splitting a double into three parts–two byes, four bytes, and two bytes, yields a two-byte value that is the majority of the data precision. The next four bytes would be most of the rest of the data precision. The last two bytes are the remainder of the precision. For quick overviews, just reading the two-byte portion is sufficient to get a rough idea of data contents. If particular data looks interesting, more and more data can be read to improve the precision. This can easily be applied uniformly by each process without coordinating. From a storage hierarchy perspective, the primary value portion could be written to high performance, low capacity storage while the rest is written to slower storage. That would reduce the data quantity written to NVM, for example, by 75% for a 2/6 split enabling far more output to be written to this fast tier with the rest spooled asynchronously.

A more complex, but interesting approach is to offer an *Auditor* that performs a similar, but lower precision computation that greatly reduces computation intensity with only a minor loss of precision over short spatial and temporal domains. As the simulation progresses, the auditor drift would need to be reset to bring it back into alignment. The much smaller, reduced precision data could be used instead of the full precision in many circumstances. A more dynamic lossy approach would be to apply a wavelet compression [7] approach that generates different compressed data sizes at a fixed error rate depending on the data entropy. These are still uniform operators, but the resulting data can be of radically different sizes. For this approach, some or all of the lower precision data can be written to the fast tier while full precision can be spooled asynchrnously to the slower tier(s).

A middle ground approach between these two refactoring approaches is to use a non-uniform operator across a variable. Instead of, for example, splitting on byte boundaries to make multiple values, a function that identifies data features through a local analytics function could identify "interesting" data that should be main-

tained in full precision while other areas can be more highly compressed. Such an operator would decompose the data from a process further than a single process or node depending on the data features. This approach may balance precision better than wavelet compression, but at a higher analytics cost. We would put data areas with interesting features on the fast tier while the rest could be spooled to the slow tier asynchronously.

We plan to investigate all three approaches as different applications will have different techniques that work best. Our initial tests have demonstrated the potential for the auditor approach in an isolated environment. We have also built a prototype ADIOS transport method for the byte-split apporach. Any performant storage system should offer the flexibility to incorporate application specific data refactoring techniques and sufficient metadata support for an arbitrary client to access the data successfully.

Quality of Service deadlines are important for driving how much time is spent performing IO. At the same time, data quality must be maintained at a sufficient level for the resulting science to be possible. The system must offer a way for an end-user to negotiate with the system for trading off between time spent in IO and the amount of data written or read. Part of that tradeoff will be increased or reduced data quality, depending on the system. For example, when data is written, a few different forms may be output given sufficient space and time. One may be every fourth element across a regular mesh reduces the data size to $1/64^{th}$ the full data set. Further dividing that using byte-splitting can reduce it by another 75%. For quick overviews to search data sets, this reduced form can be sufficient driving which data sets should be investigated in more detail. The key driving factor is time. The applicaiton scientist can read 0.4% of the full data volume with a similar time reduction. SIRIUS will offer a negotiation process to read any data.

First, a user will request a particular variable and timesetp. SIRIUS will return a table similar to Table 1. For a given variable "A", the end-user can then select the data quality and approximate reading time for the operation. In this case, there are 5 options for obtaining the whole data set (1/2/3/4, 5/6, 7/8, 9, 10). The first four represent a spatial and byte split, 5/6 represent just strided writing, 7/8 represent just a byte-split, 9 represents a wavelet compressed version, and 10 is the full precision data. The user would specify to read a particular data instance that would proceed normally. Quality of Service soft guarantees specified as part of the request will result in SIRIUS allocating soft bandwidth reservations to meet the request. While some of the error rates seem to render the data unusable, consider a 3000x3000x3000 simulation domain. That would yield a 27 gigapixel image should all of the data be used. Visualization will already radically reduce the data sizes to make usable visualizations that work on a standard display. A 99% reduction would yield a 270 megapixel image–still quite large.

Table 1: Reading Options for Variable "A"

#	Size	Time	Time Err	Refactoring	Data Err
1	$(\frac{1}{4})^3(\frac{3}{8})$A	10s	± 3s	stride, byte-split	99%
2	$(\frac{3}{4})^3(\frac{3}{8})$A	90s	± 30s	stride, byte-split	58%
3	$(\frac{1}{4})^3(\frac{5}{8})$A	16s	± 5s	stride, byte-split	0.01%
4	$(\frac{3}{4})^3(\frac{5}{8})$A	120s	± 50s	stride, byte-split	0.01%
5	$(\frac{1}{4})^3$A	1200s	± 30s	stride	98%
6	$(\frac{3}{4})^3$A	2400s	± 90s	stride	58%
7	$(\frac{3}{8})$A	1350s	± 120s	byte-split	5%
8	$(\frac{5}{8})$A	2250s	± 120s	byte-split	0.01%
9	A	36s	± 6s	wavelet	1%
10	A	3600s	± 600s	none	0%

4 Storage Challenges and Opportunities

The supercomputing platforms being deployed today are all incorporating a Non-volatile memory (NVM) layer as a fast cache between compute and the storage array. The assumption is that nearly all IO will be performed against this layer to hide the much greater latency to the storage array. The challenge is the NVM capacity is expensive limiting the amount deployed. The unspoken limitation is that this fast layer still must drain to the relatively slow storage array. This performance mismatch will limit capacity and performance just due to the drain and data stage-in operations.

A more effective way to use these resources is to only store the most important data in the fast tier and store lower precision representations of the rest of the data in the slower tiers. This lowers pressures on the NVM layer increasing performance overall by greatly reducing the drain/stage-in operations and reducing the data volumes stored on a per application basis.

5 Quality of Service Features

SIRIUS offers a novel feature related to managing IO time as a first class characteristic. Rather than being at the mercy of the storage array performance, an end-user can describe the local data to the storage system and receive an estimate on how long that data will take to write. If that is acceptable within a margin of error, the API can write. If it is unacceptable, the end-user can make decisions about skipping the output or maybe spending a little time with a strong data refactoring to reduce the data size.

On reading, a user may specify that they are willing to wait for an approximate time for an operation. This will negotiate with the storage system to determine the data quality returned. If the requested data quality cannot be retrieved within a given time, the storage array may be able to apply an additional data reduction operator to decrease the amount of time spent in IO. Alternatively, if multiple different refactored data versions are stored, alternative versions can be requested instead.

6 Metadata Challenges

The metadata system has a daunting challenge. Traditional metadata systems for both distributed and parallel file systems only have to deal with a single name space and storage hierarchy. While hierarichal storage management [1] systems offer a way to show a single name space, the reality is that the tier closest to the user is a cache for the other tiers with data migrating as necessary to support user interaction. SIRIUS is working in a fundamentally different way. Each tier is treated as a first class citizen with direct client access possible. Where data is actually retrieved from is where it is actually stored rather than forcing data migration and caching. Systems like Sirocco [5] offer a way to support this model directly and are a fundamental part of our system.

The additional challege SIRIUS metadata services face is incorporating user-specified attributes including data refactoring functions. Since an arbitrary user must be able to access any data written for which they have permissions. Having data locked behind data compression that has changed over time may result in effectively encrypting data and losing the decryption key.

We also seek to offer the ability to also store multiple versions of the same data using different refactoring techniques. This will offer additional options for higher precision OR higher performance reading.

7 Demonstration

This stack has an early prototype implementation intended to test concepts rather than performance and scalability. It has focused on examining the interaction of the different APIs for each layer to flesh out any detailed requirements or concerns that may have been missed in the conceptualization of this IO stack. To demonstrate the viability of the IO stack described in this paper, we show some very early performance results from the untuned prototype.

8 Conclusions

Overall, the SIRIUS storage system is ushering in a new generation of both IO APIs and storage system managed together. It incorporates additional metadata to help accelerate data access and ultimately time to insight. Our initial efforts show promise that such a system is both possible and can offer attractive performance to better use the supercomputing platform.

9 Acknowledgments

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

References

[1] BLAZE, M., AND ALONSO, R. Dynamic hierarchical caching for large-scale distributed file systems. In *Proceedings of the 12th International Conference on Distributed Computing Systems, Yokohama, Japan, June 9-12, 1992* (1992), IEEE Computer Society, pp. 521–528.

[2] BRAAM, P. J. The lustre storage architecture. Cluster File Systems Inc. Architecture, design, and manual for Lustre, Nov. 2002. http://www.lustre.org/docs/lustre.pdf.

[3] CARNS, P. H., LIGON III, W. B., ROSS, R. B., AND THAKUR, R. PVFS: A parallel file system for linux clusters. In *Proceedings of the 4th Annual Linux Showcase and Conference* (Atlanta, GA, Oct. 2000), USENIX Association, pp. 317–327.

[4] CHILAN, C. M., YANG, M., CHENG, A., AND ARBER, L. Parallel I/O performance study with HDF5, a scientific data package. In *14th IBM System Scientific User Group (ScicomP)* (July 2006).

[5] CURRY, M. L., WARD, L., AND DANIELSON, G. Motivation and design of the sirocco storage system, version 1.0. Tech. rep., Sandia National Laboratories, Albuquerque, New Mexico, 2015. `http://www.cs.sandia.gov/Scalable_IO/sirocco`.

[6] GHEMAWAT, S., GOBIOFF, H., AND LEUNG, S.-T. The Google file system. In *Proceedings of the NineteenthACM Symposium on Operating Systems Principles* (Bolton Landing, NY, Oct. 2003), ACM Press, pp. 96–108.

[7] KLAPPENECKER, A., AND MAY, F. U. Evolving better wavelet compression schemes. In *in Proc. of Wavelet Applications in Signal and Image Processing III, 1214* (1995), pp. 614–622.

[8] LI, J., KENG LIAO, W., CHOUDHARY, A., ROSS, R., THAKUR, R., GROPP, W., LATHAM, R., SIEGEL, A., GALLAGHER, B., AND ZINGALE, M. Parallel netCDF: A high-performance scientific I/O interface. In *Proceedings of the SC2003: High Performance Networking and Computing* (Phoenix, AZ, nov 2003), IEEE Computer Society Press.

[9] LOFSTEAD, J., POLTE, M., GIBSON, G., KLASKY, S. A., SCHWAN, K., OLDFIELD, R., AND WOLF, M. Six degrees of scientific data: Reading patters for extreme scale IO. In *Proceedings of the International Symposium on High Performance and Distributed Computing* (San Jose, CA, June 2011), ACM.

[10] LOFSTEAD, J., ZHENG, F., KLASKY, S., AND SCHWAN, K. Adaptable, metadata rich IO methods for portable high performance IO. In *Proceedings of the International Parallel and Distributed Processing Symposium* (Rome, Italy, 2009).

[11] Object-based storage architecture: Defining a new generation of storage systems built on distributed, intelligent storage devices. Panasas Inc. white paper, version 1.0, Oct. 2003. http://www.panasas.com/docs/.

[12] POWLOWSKI, B., JUSZCZAK, C., STAUBACH, P., SMITH, C., LEBEL, D., AND HITZ, D. NFS version 3: Design and implementations. In *Proceedings of the 1994 Summer USENIX Technical Conference* (1994), pp. 137–152.

[13] REW, R., AND DAVIS, G. Netcdf: an interface for scientific data access. *Computer Graphics and Applications, IEEE 10*, 4 (1990), 76–82.

[14] REW, R., SAVIS, G., EMERSON, S., DAVIES, H., HARTNETT, E., AND HEIMBINGER, D. The netcdf user's guide, data model, programming interfaces, and format for self-describing, portal data. netcdf version 4.1.3. Tech. rep., Unidata Program Center, June 2011.

[15] SCHMUCK, F., AND HASKIN, R. GPFS: A shared-disk file system for large computing clusters. In *Proceedings of the USENIX FAST '02Conference on File and Storage Technologies* (Monterey, CA, Jan. 2002), USENIX Association, pp. 231–244.

[16] SHVACHKO, K., KUANG, H., RADIA, S., AND CHANSLER, R. The hadoop distributed file system. In *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)* (Washington, DC, USA, 2010), MSST '10, IEEE Computer Society, pp. 1–10.

[17] WEIL, S. A., BRANDT, S. A., MILLER, E. L., LONG, D. D. E., AND MALTZAHN, C. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 2006Symposium on Operating Systems Design and Implementation* (2006), University of California, Santa Cruz, pp. 307–320.