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

Jay Lofstead[*], Carlos Maltzahn[†], Scott Klasky[‡], Hasan Abbasi[‡], Qing Liu[‡]
Matthew Curry[*], Mark Ainsworth[¶], Manish Parashar[§]
[*] Sandia National Laboratories [†] University of California
[‡] Oak Ridge National Laboratory [§] Rutgers University [¶] Brown University

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 exascacle applications cognizant of projected exascale platform characteristics and in the context of current industry trends.

1. INTRODUCTION

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 [8] and the end-of-life NetCDF 3.x [?]. 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 [?] and PnetCDF [?] 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 and offered a performance penalty in most cases when running at scale. Libraries like ADIOS [?] 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 [?].

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.

Systems like ADIOS showed the performance advantage for decomposed data layouts, but still rely on a uniform storage layer. SIRIUS represents a shift to consider 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 programmatic interface end users will see when interacting with the storage array. Section 4 discusses

PDSW15 November 16, 2015, Austin, TX, USA
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2505-9/13/11...$15.00.
http://dx.doi.org/xxxx.xxx/xxxxxxxxxx

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 [9] 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 [2] 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 [1] 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 [6] offers a hands-off approach for providing good performance for scaling parallel IO tasks and is used extensively by its owner, IBM. Panasas [4] 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 [3] and HDFS [7], 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 [5] that has been used for decades for enterprise file systems.

3. END-USER API LAYER

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.

Open and Close need to add additional information to include intent information. The intent hints to the system that....

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. A more dynamic approach would be to apply a wavelet compression [**?**] approach that generates different compressed data sizes at a fixed error rate depending on the data entropy. This is still a uniform operator, but the resulting data can be of radically different sizes.

A more complex, but interesting approach is to offer an *Auditor* that performs the same computation, but with some simplification that greatly reduces computation intensity with only a minor loss of precision over the short term. Longer term, this auditor would need to be reset to bring it back into alignment with the full precision simulation. The much smaller reduced precision data could be used instead of the full precision in many circumstances.

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 maintained 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 plan to investigate all three approaches as different applications will have different techniques that work best. 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.

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 [?] 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 [?] 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. ACKNOWLEDGEMENTS

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.

10. REFERENCES

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

[2] P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thakur. PVFS: A parallel file system for linux clusters. In *Proceedings of the 4th Annual Linux Showcase and Conference*, pages 317–327, Atlanta, GA, Oct. 2000. USENIX Association.

[3] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google file system. In *Proceedings of the NineteenthACM Symposium on Operating Systems Principles*, pages 96–108, Bolton Landing, NY, Oct. 2003. ACM Press.

[4] 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/.

[5] B. Powlowski, C. Juszczak, P. Staubach, C. Smith, D. Lebel, and D. Hitz. NFS version 3: Design and implementations. pages 137–152, 1994.

[6] F. Schmuck and R. Haskin. GPFS: A shared-disk file system for large computing clusters. In *Proceedings of the USENIX FAST '02Conference on File and Storage Technologies*, pages 231–244, Monterey, CA, Jan. 2002. USENIX Association.

[7] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, MSST '10, pages 1–10, Washington, DC, USA, 2010. IEEE Computer Society.

[8] The HDF Group. Hierarchical data format version 5, 2000-2014. http://www.hdfgroup.org/HDF5.

[9] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 2006Symposium on Operating Systems Design and Implementation*, pages 307–320. University of California, Santa Cruz, 2006.