Exascale Storage Systems the SIRIUS Way

S A Klasky[1,2,3], H Abbasi[1], M Ainsworth[4,1], J Choi[1], M Curry[5], T Kurc[6,1], Q Liu[1], J Lofstead[5], C Maltzahn[7], M Parashar[8], N Podhorszki[1], E Suchyta[1], F Wang[1], M Wolf[1,3], C S Chang[9], M Churchill[9], S Ethier[9]

[1] Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA
[2] Department of Electrical Engineering & Computer Science, The University of Tennessee Knoxville, Knoxville, Tennessee 37996, USA
[3] School of Computer Science, Georgia Institute of Technology, Atlanta, GA 30332, USA
[4] Division of Applied Mathematics, Brown University, Providence, RI 02912, USA
[5] Sandia National Laboratories, Albuquerque, NM 87185, USA
[6] Department of Biomedical Informatics, Stony Brook University, Stony Brook, NY 11794, USA
[7] Computer Science Department, University of California at Santa Cruz, Santa Cruz, CA 95064, USA
[8] Rutgers Discovery Informatics Institute, Rutgers University, Piscataway, NJ 08854, USA
[9] Princeton Plasma Physics Laboratory, Princeton, NJ 08543, USA

klasky@ornl.gov

Abstract. As the exascale computing age emerges, data related issues are becoming critical factors that determine how and where we do computing. Popular approaches used by traditional I/O solution and storage libraries become increasingly bottlenecked due to their assumptions about data movement, re-organization, and storage. While, new technologies, such as “burst buffers”, can help address some of the short-term performance issues, it is essential that we reexamine the underlying storage and I/O infrastructure to effectively support requirements and challenges at exascale and beyond. In this paper we present a new approach to the exascale Storage System and I/O (SSIO), which is based on allowing users to inject application knowledge into the system and leverage this knowledge to better manage, store, and access large data volumes so as to minimize the time to scientific insights. Central to our approach is the distinction between the data, metadata, and the knowledge contained therein, transferred from the user to the system by describing “utility” of data as it ages.

1. Introduction

The purpose of computing is insight, not numbers [1], a remark Richard Hamming made over 50 years ago at the dawning of the age of large scale scientific computing. The central objective of our next-generation storage and I/O system (Sirius) is to minimize the time to insight with respect to scientific computing at extreme scales. This means that we need new research into managing, storing, and retrieving the large data volumes produced by simulations, and analyzed for months afterwards.

Content from this work may be used under the terms of the Creative Commons Attribution 3.0 licence. Any further distribution of this work must maintain attribution to the author(s) and the title of the work, journal citation and DOI.
Published under licence by IOP Publishing Ltd

To begin, we present an example to illustrate these challenges and to motivate the on-going work. Next we describe our guiding principles to help tackle these challenges. In the subsequent sections we describe the key applied mathematics and computer science research components of SIRIUS. Here we introduce a new approach, data refactoring using an auditor, to aid in the separation of information from data. Finally, we describe the key challenges when this approach is realized, along with the future obstacle to manage the data lifecycle at extreme scales.

1.1. Illustrative Example

Our motivating use case is a series of simulations of the ITER fusion experiments using the XGC1 [2] application. XGC1 is one of the largest applications used at the Department of Energy's (DOE's) Leadership Class Facilities with an allocation of over 300M hours in 2015. These simulations were scheduled to produce 100 PB of data over ten days of total runtime on the Titan system at the Oak Ridge Leadership Computing Facility (OLCF) and required a team of experts, including the user group and I/O and storage personnel, to help ensure that the maximum amount of information would be saved. Due to physical resource limitations, the simulation was immediately restricted to write about 10 PB. However, when the time to write and read, together with the financial cost to archive this volume of data was fully explored, the data size goal was further reduced to only 5 PB total.

This forced a careful scrutiny of the data to be generated and restriction of the output to only the most important pieces. Further, this reduced data set had to be divided into two categories: the data that would be accessed while the simulation was running or shortly thereafter and the data that would be needed after the whole campaign was finished. The former category of data would be stored on disk and the latter would be archived. Additionally, the entire team needed to figure out how to efficiently retrieve these different categories of data for post-processing analysis and visualization.

To accomplish this, the application and storage teams developed new application specific data reduction techniques and added them to the ADIOS [3] I/O middleware layer. A rudimentary discovery system was also created to track what data was on disk and what was on tape. Initially this solution was sufficient, but eventually a new problem was discovered. Although the team had reserved compute resources for the run, the entire file system was shared, resulting in very high I/O variability. Addressing this variability, caused by contention on shared resources, is an ongoing concern. Earlier efforts [4] were insufficient due to the immense data volumes, limited time available, and the number of current users.

The problems faced by this user workflow motivate our research here. The use case demonstrated that three tiers of solutions are required: application level knowledge of data and how it will be used, middleware management of data and resources, and storage system level scheduling of resources. It also provided us with new insights. First, the users and developers of the application data can, without a great loss of encoded knowledge, reduce the size of data that needs to be stored. Second, data storage must be managed with input from the users to correctly decide the target of storage operations. Third, shared resources must be managed in a way that provides sufficient performance to all applications without encumbering the applications with high variability in performance. These insights drive the SIRIUS project and its research components.

1.2. Guiding principles of SIRIUS

The exascale storage system and I/O (SSIO) community must overcome the challenges described above for both the application scientist trying to write data and those trying to read data. This must be done such that the SSIO system can fairly share the resources among the users while helping enable exascale science by prioritizing and understanding the application level data requirements. The SIRIUS project is ultimately guided by two basic principles:

Principle 1: A knowledge-centric system design that allows user knowledge to define data policies. Today, SSIO layers are written in a stovepipe fashion and quite often do not allow optimizations. We are re-designing the layers in a highly integrated fashion where users express their intentions into the system and actions will statically and dynamically optimize for both the system and for individual requests.

Principle 2: Predictable performance and data quality in the SSIO layers need to be established to maximize the information (rather than raw data) generated on the exascale systems. Without predictable performance, runs may be slowed down because of shared resource contention, which can affect key science decisions, e.g., how much data reduction should be performed. To have some confidence that sufficient output time is available, a conservative estimate, rather than nearly accurate, can be used.

ADIOS can alleviate the need for “magic” and “tricks” to optimize application I/O performance on today's file systems by placing extra annotations in the metadata to better understand the underlying storage system [3]. In SIRIUS, we are extending this by providing a systematic autonomic approach for combining intentions and other knowledge from the *user* with performance estimations and guarantees from the underlying storage. By capturing user intentions and acting upon them in the middleware, we free the user from polluting application code with system specific optimizations. These techniques will be further integrated in combing ADIOS with an object store like RADOS-Ceph [5], which is a distributed object store and file system. Ceph offers both a POSIX and object interface including features typically found in parallel file systems.

2. Data Refactoring

The classical workflow where the entire dataset is written to storage for later analysis will no longer be viable at exascale simply because the amount of generated data will be too large due to capacity and performance limitations. In the future, it will be vital to take advantage of *a priori* information (1) to gain higher performance and predictability of I/O, (2) to prioritize the most useful data for end users so that I/O can be finished under time constraints, and (3) to perform *in situ* operations and analysis before storing the information. A result of these requirements is a need for a set of techniques to reduce and restructure data, here referred to as *data refactoring*.

The fusion use case described earlier shares commonalities with many other DOE applications. *A priori* information can be provided by application scientists regarding which data should be sent to the storage system (e.g., SIRIUS) so that minimally, the most science relevant data can be available for subsequent analysis. This allows science goals to be accomplished even when the storage is busy servicing other users. As we discussed earlier, this shared resource contention is common and causes high performance variability. For our approach, data refactoring generates the data prioritization classes. This refactoring includes data re-organization and reductions. There are many data refactoring techniques and the best choice will generally be application dependent. However, our observation is that, once the choice is settled for an application, it will not typically change from run to run within an extended campaign.

One research challenge in effectively and efficiently refactoring data is understanding when the time and resources required to identify and execute the “best methods” exceeds the gains achieved. Another critical research question concerns quantifying and controlling information loss from refactoring the data and using a reduced dataset. Broadly speaking, the path from data to knowledge consists of extracting underlying models or patterns from the datasets and interpreting the resulting models. Although scientific data generally contains random components due to finite precision and measurement and calibration effects, useful scientific data is never purely random. As such, a core concern in refactoring is understanding how much *information* is present in a dataset and therefore which type(s) of refactoring will be the most effective.

Ideally, scientists would like to perform the entire analysis *in situ,* effectively circumventing the large data issue completely. The catch, of course, is that this is unlikely to be possible since, by their nature, large-scale simulations aim to discover new information often hidden in the form of higher order effects amongst the data deluge. In particular, this means if data thinning or truncation is applied haphazardly, the higher order effects may be eliminated.

Typically, entire data sets cannot be stored in easily accessible storage due to its sheer size. Yet, the data cannot be reduced prior to archiving without risking losing information. Viewed in this way, the problem would appear intractable. As noted above, though, this is not a true impediment as long as we can incorporate a user's *a priori* knowledge of models and effective refactoring techniques. The information needed to answer the scientist's particular science goals is frequently significantly smaller, and using careful information-theoretic and application-given techniques, the SSIO layer can exploit this.

Deep application knowledge means one can sometimes achieve dramatically superior data reduction compared with what one might achieve otherwise. However, even in the absence of such high level knowledge, SIRIUS must offer generic data reduction and re-organization techniques. For instance, certain basic data semantics information is needed and must be supported by the overall infrastructure. Currently we are studying three generic and one application specific refactoring methods: (1) Precision based, (2) Frequency-based, (3) Linear Auditing, and (4) Application aware histogramming. Each of these is described below.

Precision based refactoring groups together the most significant data bits of the binary representation of each object. This data generally has a higher "utility" than data with the least significant bits, and contains less entropy. This technique can involve memory intensive operations, and as such, great care must be taken to balance the CPU, memory, and storage bandwidth costs and savings. Reading this data back introduces additional challenges—hot to find the separated components of the data if full resolution is needed and how to reconstruct the "packed" data to a form the user can work with.

Multi-resolution refactoring separates a dataset using variable granularities, or resolutions. Higher granularity samples are subtracted from lower granularity ones. This technique is commonly used in streaming data using techniques such as JPEG-2000 [6]. These mechanisms support spatial random access or region of interest access at varying degrees of granularity. In relation to this project, this capability allows us to place the lowest granularity chunks in the fastest storage and the highest granularity chunks can either be written to slower tiers or not written at all. In order to fully take advantage of multi-resolution re-organization, techniques to pre-condition the data can be used to further optimize this process. This technique can be applied together with Precision-based refactoring methods as the pre-conditioner.

Linear Reconstruction is a "technique" for reducing data consists of storing every s data item and simply ignoring the rest of the data. This approach is frequently adopted as a simple, but often effective method for data reduction. The tactic assumption underlying the approach is that the discarded values could, if required, be reconstructed by interpolating the retained values. More generally, one can retain every s-th item of data, where s is a chosen stride length resulting in an s-fold reduction in the size of the data. Objections to this and related ad hoc procedures are their lossy nature and lack of a sound theoretical basis. In order to address the lossy nature, we are using the linear reconstruction procedure described above as a predictor for the discarded items of data. The difference, or deltas, between the actual value of the discarded item can be shown, under reasonable assumptions, to have a significantly lower entropy than the original data. This means that the deltas are amenable to compression using, for instance, a standard arithmetic encoder. The resulting procedure then becomes lossless. The process of reinstating the data consists of identifying the stored values at the adjacent strides s, performing linear interpolation to obtain a predicted value, using the predicted value as key to access the delta correction from the arithmetic encoder, and finally, summing the predicted and delta corrector to obtain the original data item.

The resulting procedure can be surprisingly effective. Figure 1 shows the results obtained for varying choices of stride length s for the storage of data sampled from the function $sin(x)\ exp(20x)$, $0<x<1$ at uniformly spaced points. The compression ratio is computed by comparing the number of bits that would be required to store the original, full data set with the number of bits required to store the compressed data set including the costs (and overhead) associated with the arithmetic encoder.

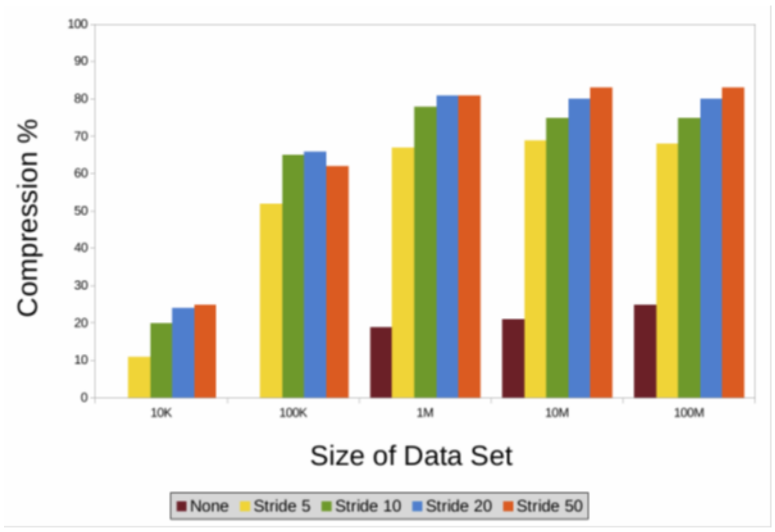

Figure 1. Compression ratio obtained using linear reconstruction for different choices of string length s.

Remarkably, the optimal choice for the stride s is considerably larger than the choice *s=10* often used in practice despite the fact that the underlying data has a variation of several orders of magnitude across the sampling range.

Application Aware Auditor contains a class of techniques which have more domain knowledge than the generic techniques described above. In our example, physicist's understands that the particle distribution closely follows a Maxwellian distribution [7]. By using methods borrowed from perturbation theory[10] we can remove the background Maxwellian distribution and then store only the reduced non-Maxwellian piece of the distribution function. Physically, this should have a much smaller dynamic range. In our example we applied this technique using a two dimensional histogram of the particles in velocity space, and were able to get a dramatic reduction of the data to be saved using this lossy technique. This technique can be further combined with our other refactoring techniques to further reduce the potential storage cost of the information.

3. Refactoring Challenges

Refactoring large volumes of data is a challenging computing problem with three broad challenges. (1) Large volumes of data need to be refactored with *minimal impact* on the *application* and the computational *resources*. (2**)** The refactored data has to be created in a manner consistent with the *access patterns of the data consumer.* This requires predicting the common access patterns for data sets, identifying the most likely patterns and then correctly selecting the appropriate refactoring code path. (3) The abstraction through which data is viewed by the user needs to be *consistent in the presence of changing refactoring techniques* (in order to minimize the cognitive burden on the user), while providing sufficient transparency to enable both user and system to optimize metrics such as data layout, data accuracy and predictable performance. Next we describe these broad areas in detail and elaborate our approach for addressing these key challenges.

3.1. Resource Utilization

The generation of large volumes of data, is a costly and resource intensive apparatus. Additional work required by data refactoring can only be justified if the benefits outweigh the costs, and if the overall impact on the performance of the data pipeline is positive. Thus, any additional task to refactor,

[10] See e.g. https://en.wikipedia.org/wiki/Perturbation_theory

reorganize or compress data will need to account positively for the consumption of resources. In particular, there are three resources that are critical to scientific big data pipelines, on board system memory, additional time on the processor, and bandwidth consumed for data movement.

If we consider memory as the first constraint, data refactoring requires maintaining an original baseline state in memory, while generating the pieces to represent the refactored state. In the precision based refactoring technique, the original data is kept in memory while tiered precision arrays are generated. This requires that we double the memory requirement, and in some, very memory constrained use cases, this might render the refactoring an unviable approach. To overcome this obstacle, we have developed a method to refactor data streams by utilizing temporary windows over a stream of data. This windowed approach limits the additional memory overhead, but incurs a performance penalty when outputting data. Thus, we are studying optimal window sizes for the different refactoring techniques we have described earlier.

Similarly, for CPU and bandwidth consumption, the consumed resources will have an impact on the performance of the application. Some of this can be mitigated by utilizing asynchronous methods to compute and move the refactored data. This will require careful management of when the refactoring computation is called, and when data is moved, to minimize the overhead on the application due to resource contention. Here we are extending our past work on contention avoidance for data movement and resource sharing [8].

3.2. Refactoring Selection
There exist multiple techniques to refactor data that provide a varying set of tradeoffs between data output and access times, storage utilization, and accuracy of results. The selection of the refactoring technique becomes particularly challenging due to two factors. First, the requirements from the users evolve over periods of time, as the importance and utility of data change. Secondly, refactoring techniques can be used in combination, in any non-specific order.

Appropriately selecting the set and ordering of refactoring methods will require evaluations of the strengths of each combination within specific user constraints. The combination of user needs and data characteristics will require an autonomic controller to select the refactoring method. For example, a delta compression such as using the Linear Auditor will benefit greatly from an initial refactoring into multiple precision bins (since minor noise in the bins for higher order bits will be smoothed out). However, if the user requirement is that all data must be accessed at maximum precision, the additional work for precision based refactoring is unnecessary.

3.3. Linked Data Abstraction
Separate from the resource and performance impact of refactoring on the data intensive application, is the increased complexity of the organization and structure of the stored data. To be able to provide users with a compelling case for adopting this approach, a succinct data abstraction is required. This data abstraction must be able to provide a coherent view of discrete blocks of data, with each block having passed through multiple refactoring functions. Moreover, as the system makes decision on moving the various blocks based on importance and utility, the abstraction must be able to encapsulate the location of the blocks and allow a common API to access the data transparently. We are building this abstraction on the current ADIOS data abstraction which creates separate blocks for each process in a parallel cohort. We will extend this with deep links and attributes that describe the type and parameters of the refactoring operation. Achieving this combination of transparent data access with a performant middleware system will be one of the key challenges that will need to be solved to provide a usable data platform for exascale applications.

4. Managing Data Lifecycle
SIRIUS aims to manage the overall data life cycle, including data generation (e.g., from a simulation) or acquisition (e.g., in the case of experimental and observational data), optimized data placement, runtime data management including migration, reorganization and reduction, data consumption for

knowledge discovery, and purging data from the system to optimize system operation. Key research questions addressed by SIRIUS include: (1) How can we initially place data so that it can be discovered and consumed efficiently? (2) How can the placement and migration of data across a multi-tiered storage hierarchy be optimized at runtime, both from the application and system perspective? (3) How can knowledge about the application used to better prepare the data for consumption? (4) When and how do we make the decision to purge data?

Data placement and movement: When an application outputs or accesses data, the storage and middleware layers needs to decide what data is placed where in the multi-level storage system. This placement decision can have a significant impact on data management throughout the lifetime of the data. For example, our past work on data–staging [[9],[10] on HPC systems with multi-level memory structures has shown that different output techniques targeting different layers of the storage hierarchy can have a significant impact on the overhead observed by the application for I/O operations. A key requirement is application-driven runtime mechanisms for dynamically managing data placement across the layers of the distributed storage hierarchy throughout the data lifecycle, coordinating data movement and data sharing between the components of the application workflow, with the overarching goal of maximizing the relative utility to the application as well as the system while reducing access costs. As noted before, the complexities of heterogeneous multi-level storage structures require adaptive placement policies are required to be implemented to optimally utilize storage resources vertically (across deep memory hierarchies) and horizontally (across nodes within a memory level) while accommodating dynamic application requirements and transient system states. Our approach is to increase knowledge about the data and its use within the application and leverage this knowledge to drive data placement and overall management.

Soliciting application hints: As described in the previous section, one of the main components of our proposed storage system is the ability to reorganize, refactor, and reduce data as it is generated and to reorganize and possibly regenerate the data as it is accessed. We carry this principle into the placement and movement of data by allowing applications to define hints and policies that guide what data is placed where. We will explore the use of application hints in two distinct areas. First, we will study the challenges and trade-offs of either augmenting the I/O interface with hints or allowing the addition of an external specification that defines the use case. Our experience with developing modern I/O interfaces has shown that both techniques have value [9] and we will investigate the set of hints that are embedded in the application code vs. those that are described within a non-compiled specification. Second, we will study how hints can guide data placement as data is handed off from application to storage (during a write) and from storage to application (during the read). In both cases we will study what minimal set of annotations and hints can allow the storage system to minimize data movement and optimize the resources consumed by I/O.

5. Metadata Challenges

The challenge when applying refactoring techniques, particularly application aware techniques, is how to incorporate sufficient knowledge in the storage system such that an arbitrary future client has sufficient information to recreate the desired information. Additionally, by spreading data across multiple different kinds of storage media that typically have independent namespaces, locating any particular data will be challenging.

SIRIUS will provide sufficient built-in and extensible metadata services to support efficient data access. First, detailed data metadata, such as array dimensions and other similar data, must be visible within the metadata for any effective data selection. Second, each data chunk stored within the storage hierarchy must have some way to address it. As data utility forces data migration within SIRIUS, the metadata must either dynamically track data as it moves or offer a search feature to discover where data currently resides. Which approach is superior and under what conditions still needs to be determined through further research.

6. Conclusion

Extreme scale application workflows, such as the fusion workflow described in this paper, generate very large amounts of data that need to be processed and analyzed before potential insights from the simulations can be realized. Managing, storing and retrieving these large volumes of data have become critical challenges. The central objective of the SIRIUS project is to address these challenges and minimize the *time to insight* for scientific workflows at extreme scales by enabling predictable performance across the storage system. We are creating a new understanding of how users can describe their intentions by describing the data utility, and refactor their data to achieve faster storage, retrieval, and understanding of the information contained in the data.

Acknowledgments
This project is funded by the following research grants from the Advanced Scientific Research Office at the Department of Energy, under contracts SDAV: ERKJ200/KJ0403000, RSVP: ERKJU60/KJ0402000, and Sirius: ERKJ311/KJ0402000. 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] Hamming R W 1986 *Numerical Methods for Scientists and Engineers* 2nd Ed. (New York: Dover)
[2] Chang S *et al.* 2006 Integrated particle simulation of neoclassical and turbulence physics in the tokamak pedestal/edge region using XGC *FEC 2016 Proc* (Chengdu: IAEA) p 119
[3] Liu Q *et al.* 2014 Hello ADIOS: the challenges and lessons of developing leadership class I/O frameworks *Concurr Comput* ***26*** pp 1453–1473.
[4] Lofstead J, Zheng F, Liu Q, Klasky S, Oldfield S, Kordenbrock T, Schwan K and Wolf M 2010 Managing Variability in the IO Performance of Petascale Storage Systems *SC12 Proc* (New Orleans: IEEE) pp 1-12
[5] Weil S A, Brandt S A, Miller E L, Long D D E and Maltzahn C 2006 Ceph: a scalable, high-performance distributed file system *OSDI 2006 Proc* (Berkeley: USENIX) pp 307-320
[6] Marcellin M W *et al*. 2000 An overview of JPEG-2000 *DCC 2000 Proc.* (Snowbird: IEEE)
[7] Krook M and Wu TT 1976 Formation of Maxwellian tails *PRL* **36** p 1107
[8] Abbasi H *et al.* (2010) Datastager: scalable data staging services for petascale applications *Cluster Comput* **13** pp 277-290
[9] Sun Q, Zhang F, Jin T, Bui H, Romanus M, Yu H, Kolla H, Chen J and Parashar M 2015 Adaptive Data Placement For Staging-based Coupled Scientific Workflows *SC15 Proc (Austin: IEEE)*
[10] Tong J *et al.* 2015 Exploring Data Staging Across Deep Memory Hierarchies for Coupled Data Intensive Simulation Workflows *IPDPS 2015 Proc* (Hyderabad: IEEE)