EMPRESS—Extensible Metadata PRovider for Extreme-scale Scientific Simulations

Margaret Lawson*
Sandia National Laboratories
Albuquerque, New Mexico
mlawso@sandia.gov

Jay Lofstead, Scott Levy, Patrick Widener
Sandia National Laboratories
Albuquerque, New Mexico
gflofst|sllevy|pwidene@sandia.gov

Craig Ulmer, Shyamali Mukherjee, Gary Templet
Sandia National Laboratories
Livermore, California
cdulmer|smukher|gjtempl@sandia.gov

Todd Kordenbrock
DXC Technology
thkorde@sandia.gov

ABSTRACT

Significant challenges exist in the efficient retrieval of data from extreme-scale simulations. An important and evolving method of addressing these challenges is application-level metadata management. Historically, HDF5 and NetCDF have eased data retrieval by offering rudimentary attribute capabilities that provide basic metadata. ADIOS simplified data retrieval by utilizing metadata for each process' data. EMPRESS provides a simple example of the next step in this evolution by integrating per-process metadata with the storage system itself, making it more broadly useful than single file or application formats. Additionally, it allows for more robust and customizable metadata.

KEYWORDS

metadata, custom metadata, SIRIUS, Decaf, ATDM, data tagging

ACM Reference Format:
Margaret Lawson, Jay Lofstead, Scott Levy, Patrick Widener, Craig Ulmer, Shyamali Mukherjee, Gary Templet, and Todd Kordenbrock. 2017. EMPRESS—Extensible Metadata PRovider for Extreme-scale Scientific Simulations. In *PDSW-DISCS'17: PDSW-DISCS'17: Second Joint International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems, November 12–17, 2017, Denver, CO, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3149393.3149403

1 INTRODUCTION

Aided by petascale computing, scientific simulations are generating increasingly large data quantities. For example, production fusion reactor simulations like XGC1 [3] can generate 256 MiB/process at O(1M) processes with an output frequency of at least every 15 minutes over a 24 hour run, if not faster. Other data collectors, such as the Square Kilometer Array Radio Telescope (SKA), will generate as much as 20 TB/sec when operating at full functionality [1]. With flash-based solid state storage and other technologies either in production use or close on the horizon, IO bandwidth limitations are becoming less critical and the problems are shifting to finding the right data in the vast seas stored. It is increasingly important to improve techniques to quickly locate interesting data subsets without having to search data after it is stored. For example, one of the features of interest for plasma in a fusion reactor is technically called a 'blob'. These are areas of turbulence that may be one of the sources of instability that prevent a production level fusion reactor from being stable. Identifying these areas and being able to find them quickly after the simulation run is complete will accelerate the time to understanding. For the SKA, labeling different astronomical phenomena that have been discovered in a composite image will aid astronomers in selecting which images to view or analyze rather than requiring them to do the analysis after the fact delaying their work. Some solutions to this problem exist, but none work well enough to be a general solution employed at large scale by computational scientists.

The ASCR SIRIUS project is addressing this gap through several efforts. The project provides predictable storage performance thereby giving scientists the ability to adjust their work based on the time an operation will likely take. The project also includes extensible, custom metadata (provided by EMPRESS) that accelerates data selection by eliminating the need to read and scan all data and offers the potential to migrate to different storage locations based on a utility metric. Also, prior to an analysis session, SIRIUS uses data placement services to spread data across multiple storage tiers based on data utility and the desired operation. And finally, SIRIUS incorporates data reduction techniques with varying, generally minimal, information loss to aid the utility decisions.

Existing solutions to data discovery, such as HDF5 and NetCDF, eased data retrieval by offering rudimentary attribute capabilities that provide basic metadata. ADIOS [6] included a first attempt at EMPRESS-like data tagging, but suffered due to scalability limitations and overly compact storage. Other recent efforts include (FastBit) [12], Indexing [13], (AMR querying) [14], and Kilmatic [9], but these suffered from similar issues.

Empress offers two primary contributions. First, it offers fully custom, user created metadata that can be added to either a data chunk (i.e., the part a single process produced/owns) or a variable in its entirety. Second, rather than just offering a standard POSIX-style interface to read a variable or dataset, EMPRESS has described a

*Also with Dartmouth College.

PDSW-DISCS'17, November 12–17, 2017, Denver, CO, USA

ACM ISBN 978-1-4503-5134-8/17/11…$15.00
https://doi.org/10.1145/3149393.3149403

new data query interface focused on data contents rather than the highest-level metadata (i.e., dataset number and variable name + a hyperslab). This first EMPRESS prototype extends the scalable storage system with a SQLite [8] in-memory relational database to ease data searching. Future work will investigate other data storage strategies in addition to many other unanswered questions.

The rest of the paper is organized as follows. Section 2 contains an overview of the design elements and decisions. Section 3 contains the testing and evaluation information for this first iteration of EMPRESS. Next, Section 4 is a discussion of how this met our goals. Finally, Section 5 contains remarks about the next steps for the development of EMPRESS.

2 DESIGN

Empress is built using the Faodail [10] infrastructure. Faodail offers data management services initially conceived to support Asynchronous Many-Tasks (AMT) applications, but has since been generalized to support broader independent service and workflow integration tasks. For this project, the data management services are being augmented with rich metadata functionality. The base Faodail metadata is based on key-value stores driving everything through keys. Faodail is built upon the long stable and performant NNTI RDMA communication layer from the Nessie [7] RPC library from Sandia. All of the layers above the communication management have been replaced offering better C++ integration and richer functionality.

The initial design point being explored is a series of fully independent metadata servers that each store a portion of the metadata generated during a particular output operation. Later reading requires that an interface layer like ADIOS queries some number of these servers to find all of the relevant data pieces. To manage consistency, EMPRESS employs the D^2T [4] system to manage the metadata and thereby ensure consistency and availability. One might consider a key-value store like MDHIM [2] for this storage, but it is unlikely efficient enough. Consider searching for all chunks within a bounding box that does not fall on the same boundaries encoded in the key. Converting text to numbers and comparing is radically slower than using an indexed relational table column.

The process of writing to or reading from the metadata servers is broken down into two parts: the client side, which makes requests, and the server side, which stores in the local database and executes the requests. Clients and servers are both run in parallel, and are run on separate nodes representing a shared service on an HPC platform. For each broad category of interaction (e.g., creating a variable or custom metadata class, inserting a chunk or custom metadata instance, or reading all the custom metadata for a chunk) there is a client-side function. Each function sends a message with all the needed information to the selected server.

The back-end data storage layer being targeted is the RADOS layer of Ceph [11]. We link the object IDs and locations from RADOS in the metadata to reveal the exact data that should be read based on a particular metadata operation.

2.1 Data Storage Technique

For this initial effort, each metadata server maintains a SQLite [8] in-memory database and provides basic capabilities for adding custom, arbitrary metadata tags and retrieving data based on these tags. The

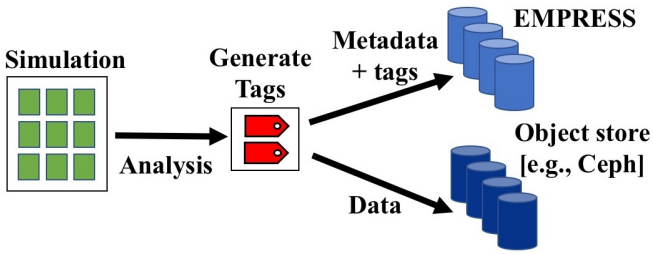

Figure 1: Targeted Workflow for Writing

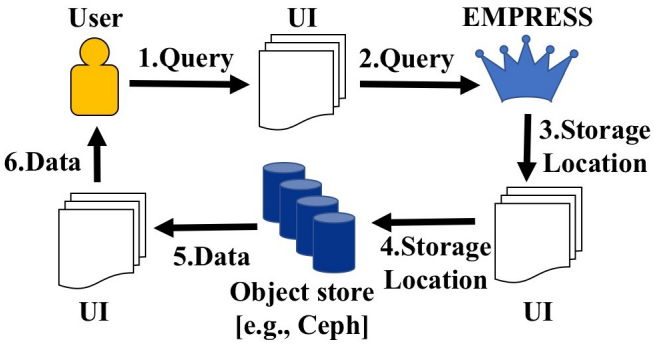

Figure 2: Targeted Workflow for Reading

proper dividing line for a metadata system for exascale computing is not clear. We expect O(10 Trillion) objects in a production exascale system EMPRESS is intended to support. Not only is this too large for a single database, but it is very large for any existing key-value system. The first step will likely use the metadata servers to host in-memory databases for currently open data sets.

2.2 Tracked Metadata

This first version of EMPRESS tracks four different broad categories of metadata: variable information, variable chunk information (i.e., the portion of a variable from a single process), custom metadata classes, and custom metadata instances.

2.2.1 Variable Information. A category of data measured for a simulation, e.g., temperature or speed. Each variable entry can store an id, a name, a path (similar to the path group hierarchy in HDF5), a version (in case the simulation recurses on an area using different granularity), whether the variable is active (for transaction purposes), dataset id, number of dimensions, and the global dimensions of the simulation space. Variables serve as a catalog for the types of data stored in the database.

2.2.2 Variable Chunk Information. A subdivision of the simulation associated with a particular variable. Each chunk stores an id, its associated variable id, the storage location of its associated data, the size of the data in bytes, and the offset of the subdivision. For example, one part of the global temperature variable covering x:(0-199), y:(1000-1199), z:(400-599).

2.2.3 Custom Metadata Class. A category of metadata that the user adds for a particular dataset. Each class entry stores an id, name, version, whether the class is active (for transaction purposes), and

its associated dataset number. Class entries serve as a catalog for the types of custom metadata instances found in the database. For example, a new metadata class might be a flag, bounding box for a region, max value for the chunk or variable, or min value for the chunk or variable.

2.2.4 Custom Metadata Instance (attribute). A piece of metadata that is associated with a custom metadata class and a specific chunk and therefore a particular variable. For each instance, the following are stored: instance id, associated chunk id, associated type id, and the metadata value. For example, max = 10 for a chunk that stores temperature data and represents a chunk in the global space with dims x:(0-199), y:(1000-1199), z:(400-599) or flag = true for a chunk that stores the x component of a velocity and has dims x:(200-399), y:(400-599), z:(1400-1599).

2.3 Interaction Examples

The proposed interaction approach is more database-like rather than POSIX API. Below, both the writing and reading API are presented.

2.3.1 Writing Example. Algorithm 1 demonstrates the basic write process. Each client writes a different portion of all variables and custom metadata classes, ensuring that in the end each server has a copy of all of them. Each client then inserts a var chunk per variable and the desired custom metadata instances for that var chunk, and through this process there are chunks written for the entire simulation space for each variable. One obvious optimization that could be made is to offer a way to insert a list of chunks and a list of custom metadata type instances for these chunks. Missing from this example is the actual data storage portion itself. Sirius currently is using Ceph's RADOS object store and data storage would use the RADOS API. The current API is most similar to ADIOS. Since this is intended to live behind an IO API like ADIOS or HDF5, separate calls for the metadata and the data storage is deemed acceptable—particularly for this early investigation.

Algorithm 1 Writing algorithm

```
1:  procedure WriteTimestep                    ▷ Each process does this
2:      for all variables assigned do
3:          md_create_var (...)
4:      end for                                ▷ Write portion of all vars
5:      for all custom metadata classes assigned do
6:          md_create_type (...)
7:      end for                ▷ Write portion of all custom md types
8:      for all variables do
9:          md_insert_chunk (...)     ▷ Add a var chunk; get the ID
10:         for all custom metadata desired do
11:             md_insert_attribute (...) ▷ Add custom md instance
12:         end for
13:     end for
14: end procedure
```

2.3.2 Reading Example. The basic reading example is presented in Algorithm 2. The initial implementation assumes that searching multiple metadata server sources for the complete list of chunks of interest is lower server overhead than having the servers coordinate. While this may impose more load at large scale, intelligently splitting queries by clients and then having clients coordinate can greatly reduce the load for both the clients and the servers.

Algorithm 2 Reading algorithm

```
1:  procedure ReadData                         ▷ Each Process Does this
2:      md_catalog_vars (...)      ▷ Get list of vars from any server
3:      for all metadata servers needed do
4:          md_get_chunk(...)   ▷ get all chunks in area of interest
5:          for all chunks returned do
6:              md_get_attribute (...) ▷ get the custom md instances
7:          end for
8:      end for
9:  end procedure
```

3 EVALUATION

Testing is performed on the Serrano capacity cluster at Sandia. Serrano has 1122 nodes with 36 cores/node (2 sockets each with 18 cores running 2.1 Ghz Intel Broadwell processors). It has an Omni-path interconnect and has 3.5 GB RAM per node. The software environment is the RHEL7 using the GNU C++ compiler version 4.9.2 and OpenMPI 1.10. Additional tests are also run on the Chama and Sky Bridge capacity clusters at Sandia. Those show similar results and are omitted for space considerations.

We perform 5 tests for each combination of the clients and servers that have a client to server ratio of at least 32:1. We use 1024 or 2048 client processes; 16, 32, or 64 server processes; and either 0 or 10 custom metadata classes per dataset.

3.1 Goals

The evaluation offers a proof of concept that examines the effect of client to server ratios on writing and reading metadata, offers some raw performance numbers to determine if the approach may be scalable for exascale-sized workloads, and demonstrates the data query interface is capable of supporting common analysis reading operations. Finally, it seeks to quantify the overhead of including large numbers of custom metadata items that can accelerate later data searching.

Each test uses two datasets with 10 globally distributed 3-D arrays per dataset. A dataset is the entirety of a single, parallel output operation, such as an analysis or checkpoint output. For this initial test, this is sufficient to show that the approach can support more than a single dataset. Scalability performance testing depends on many more variables than just the number of datasets per metadata instance and is beyond the scope of this paper.

All testing configurations that include custom metadata classes have the same 10 classes per dataset in terms of datatype for each class, name, and % of chunks containing a metadata instance of that type. Data types for these are double (max and min), blob bounding box (made up of 2 points, each made up of 3 doubles), a Boolean (flag), and two integers (a range of values). The frequencies for these range from 100% for max and min while the rest range from .1% to 25% of chunks.

The categories and frequencies of custom metadata classes, for the combinations that include custom metadata classes, are shown in Figure 3. Note: on average there are 2.641 custom metadata instances per chunk. These metadata classes and frequencies were chosen to estimate normal use. More classes and higher frequencies can easily be supported.

Class	Data Type	Frequency (% of all chunks)
Max	Double	100%
Min	Double	100%
Bounding Box 0	2 points (each made up of 3 doubles)	20%
Bounding Box 1	2 points (each made up of 3 doubles)	2.5%
Bounding Box 2	2 points (each made up of 3 doubles)	.5%
Flag 0	Boolean	25%
Flag 1	Boolean	5%
Flag 2	Boolean	.1%
Range 0	2 integers	10%
Range 1	2 integers	1%

Figure 3: Various custom metadata items used

A minimum of 5 runs of each configuration are performed. At the end of the testing, a client gathers all of the timing information and outputs it to a file. A server does the same, also outputting information about the storage sizes of the SQLite3 databases. Each time point is recorded using a high resolution clock that outputs the current time to the nanosecond. The distribution of custom metadata instances and their data values were randomized using a random seed generator with the seed being the client's rank.

3.2 Process

Each subsequent subsection describes how a different phase of the testing is performed.

3.2.1 Writing Phase. Each client is assigned to a server based on its rank, thereby ensuring as close to an even number of clients per server as possible. Then, all of the clients assigned to a given server evenly split the task of inserting the variables and custom metadata classes, thus ensuring that each server has a copy of each variable and custom metadata class. Each client is then assigned a unique subdivision of the simulation space based on its rank, and inserts all chunks for this space (one chunk per variable per dataset) and its associated custom metadata instances (2.641 attributes per chunk on average).

3.2.2 Reading Phase . Only 10% of the write-clients are used to read. Each read-client starts by connecting to each server since data is left distributed across the servers. Because each server has a copy of the variables and custom metadata classes for each dataset, all queries related solely to these are performed by querying only one server. For example, retrieving a catalog of the variables or custom metadata classes stored for a given dataset. All queries related to chunks or attributes must be sent to all servers since any chunk and its associated attributes could be on any one of the servers.

In the reading phase, metadata is read in six different patterns that scientists are likely to use frequently as identified in the Six Degrees of Scientific Data [5] paper. The six patterns are: retrieving all chunks for the given dataset, retrieving all chunks for one variable for the given dataset, retrieving all chunks for three variables for the given dataset, retrieving all chunks intersecting a plane (performed in each dimension) for the given dataset, retrieving all chunks in a subspace of the simulation space for the given dataset, and retrieving all chunks intersecting a partial plane (performed in each dimension) for the given dataset. Additionally, if the testing configuration includes custom metadata classes, the clients perform the six read patterns again on the same datasets, but after retrieving the chunks they also retrieve all of the custom metadata instances associated with these chunks.

3.3 Results

Figure 4 demonstrates the breakdown of the average operation time for retrieving all of the chunks in a given area (e.g., a plane or a subspace of the simulation space) for the 2048 client and 32 server tests with 10 custom metadata classes on Serrano. This is representative of the breakdowns for the other ops. The graph demonstrates that, at this stage, almost all function time (82%) is spent waiting for the server to begin processing the client's message. The small percentage of time spent reading from the database (11% spent on the server's interaction with the database) indicates that EMPRESS' database system and querying style are relatively efficient. Future steps will include exploring ways to reduce the server response latency and thereby greatly increase the speed of operations. These results are presented abstractly as percentages rather than actual timings to afford applying these percentages to extrapolate using other techniques for different portions of the operation execution process.

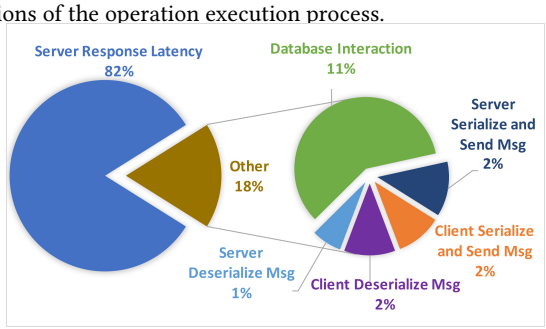

Figure 4: Breakdown of chunk retrieval time

Figure 5 demonstrates how average read and write time varies across the different configurations run on Serrano, which is representative of the clusters. This graphs provides important insight about the scalability of EMPRESS in its current form and about the impact of adding custom metadata instances to read and write queries. As indicated in the graphs, when the ratio of clients to servers is held constant, there is generally either no penalty to having more clients and servers, or there is in fact an increase in performance. This provides promising evidence that EMPRESS will be able to maintain similar performance with larger quantities and clients and servers. As expected, the ratio of clients to servers greatly affects performance. As indicated in Figure 4, almost all operation time is spent waiting for the server to act on the client's message. Thus, with fewer servers to respond to client's messages, performance is reduced. Finally, the graph demonstrates that adding custom metadata classes and instances performs as expected by the increase in data written and queried. On the write side, adding custom metadata means that in addition to having 20 variables (10 per dataset, 2 datasets) and (number of clients * 20) chunks (one

chunk per client per variable), there are 20 custom metadata classes (10 per dataset, 2 datasets) and around (2.64 * number of clients) custom metadata instances. This is thus around 2.5 times as much data to write and insert, and around 2.5 times as many client-server operations. Thus, EMPRESS performs remarkably well with the addition of custom metadata, particularly on the read side.

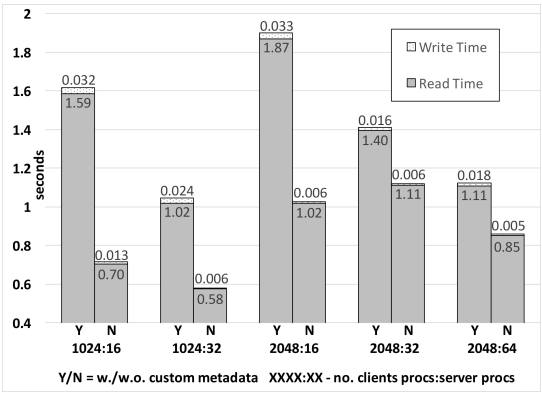

Figure 5: Time spent in reading and writing operations

Figure 6 demonstrates the client-server operation times for the 7 operations used in the write and read phase for the tests with 2048 clients, 32 servers, and 10 custom metadata classes performed on Serrano. These runs are relatively representative of the other configurations and clusters, excepting that average operation time varies with the ratio of clients to servers as discussed above. The four operations used in the write phase are writing variables ("Create Var"), chunks ("Insert Chunk"), custom metadata classes ("Create CMC"), and custom metadata instances ("Insert CMI"). The 3 operations used in the reading phase are retrieving the catalog of variables ("Catalog Var"), retrieving the chunks found in a given area ("Get Chunks"), and retrieving the custom metadata instances for a given chunk ("Get CMI"). These times show that EMPRESS can efficiently support a wide range of queries and custom metadata. In particular, custom metadata instances are retrieved and inserted in around the same amount of time as chunks, and custom metadata classes are inserted into the database even more quickly than variables. Unsurprisingly, the longest operations are retrieving the entire variables catalog, retrieving all of the chunks for a given area, and retrieving all of the custom metadata instances for a given chunk. All of these require the server to read through its database for all variable, chunk, and custom metadata instance entries respectively. Even so, with two datasets each of these operations only takes a couple of milliseconds.

4 DISCUSSION

Testing shows that EMPRESS reliably and efficiently supports a wide variety of operations including custom metadata operations. This demonstrates that a metadata service like EMPRESS can be used to provide tools for more efficient data discovery with the use of rich fully customizable metadata and database-like queries. In particular, testing proves that EMPRESS can efficiently support fully custom metadata, with custom metadata operations taking the same amount of time or less than their counterparts. Testing shows that

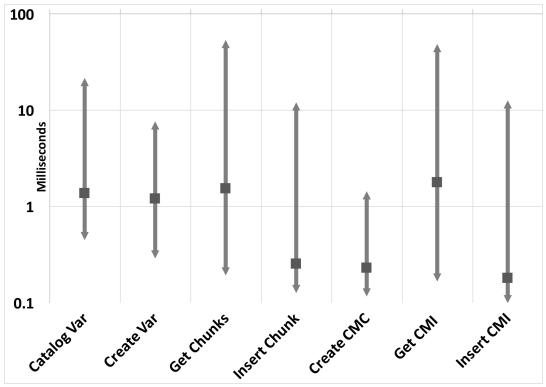

Figure 6: Operation duration distributions

the total impact of writing and reading custom metadata is equal to or less than what is expected based on the number of client-server operations and data size. In addition, there is significant room for improvement. Currently, only 10% of the total time for reading and writing operations is spent performing the database operations on average. The largest portion of an operation's time (around 80% on average) is accounted for by latency of server response. There are many options for reducing this latency time that will be explored in the future.

5 CONCLUSION AND FUTURE WORK

EMPRESS is in its nascent stages. Focus for future development includes having servers write to a storage system after each output iteration thereby handling larger datasets and a larger quantity of datasets. This will make EMPRESS more flexible in the number and variety of servers it uses for writing vs reading. We will also explore different methods of distributing the metadata across the servers. We plan to make the data values stored in custom metadata instances queryable. We will also look at further expanding custom metadata capabilities and the variety of database queries supported. This future work will be focused on increasing EMPRESS' flexibility, efficiency, and scalability. Short term plans for future testing include using strong scaling, comparing this metadata system to fixed-file metadata storage, testing custom metadata classes and instances scalability limits, and increasing the number of clients and servers.

ACKNOWLEDGEMENTS

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

This work was supported under the U.S. Department of Energy National Nuclear Security Agency ATDM project funding. This work was also supported by the U.S. Department of Energy Office of Science, under the SSIO grant series, SIRIUS project and the Data Management grant series, Decaf project, program manager Lucy Nowell.

REFERENCES

[1] Peter E Dewdney, Peter J Hall, Richard T Schilizzi, and T Joseph LW Lazio. 2009. The square kilometre array. *Proc. IEEE* 97, 8 (2009), 1482–1496.

[2] Hugh Greenberg, John Bent, and Gary Grider. 2015. MDHIM: A Parallel Key/Value Framework for HPC.. In *HotStorage*.

[3] S. Ku, C. S. Chang, M. Adams, E. D Azevedo, Y. Chen, P. Diamond, L. Greengard, T. S. Hahm, Z. Lin, S. Parker, H. Weitzner, P. Worley, and D. Zorin. 2008. Core and Edge full-f ITG turbulence with self-consistent neoclassical and mean flow dynamics using a real geometry particle code XGC1. In *Proceedings of the 22th International Conference on Plasma Physics and Controlled Nuclear Fusion Research*. Geneva, Switzerland.

[4] Jay Lofstead, Jai Dayal, Karsten Schwan, and Ron Oldfield. 2012. D2t: Doubly distributed transactions for high performance and distributed computing. In *Cluster Computing (CLUSTER), 2012 IEEE International Conference on*. IEEE, 90–98.

[5] Jay Lofstead, Milo Polte, Garth Gibson, Scott Klasky, Karsten Schwan, Ron Oldfield, Matthew Wolf, and Qing Liu. 2011. Six Degrees of Scientific Data: Reading Patterns for Extreme Scale Science IO. In *In Proceedings of High Performance and Distributed Computing*.

[6] Jay Lofstead, Fang Zheng, Scott Klasky, and Karsten Schwan. 2009. Adaptable, metadata rich IO methods for portable high performance IO. In *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*. IEEE, 1–10.

[7] Ron A. Oldfield, Patrick Widener, Arthur B. Maccabe, Lee Ward, and Todd Kordenbrock. 2006. Efficient Data-Movement for Lightweight I/O. In *Proceedings of the 2006 International Workshop on High Performance I/O Techniques and Deployment of Very Large Scale I/O Systems*. Barcelona, Spain.

[8] Michael Owens. 2003. Embedding an SQL database with SQLite. *Linux Journal* 2003, 110 (2003), 2.

[9] Tyler J Skluzacek, Kyle Chard, and Ian Foster. 2016. Klimatic: a virtual data lake for harvesting and distribution of geospatial data. In *Parallel Data Storage and data Intensive Scalable Computing Systems (PDSW-DISCS), 2016 1st Joint International Workshop on*. IEEE, 31–36.

[10] Craig Ulmer, Shyamali Mukherjee, Gary Templet, Scott Levy, Jay Lofstead, Patrick Widener, Todd Kordenbrock, and Margaret Lawson. 2017. Faodail: Enabling In Situ Analytics for Next-Generation Systems. In *Proceedings of the 2017 In Situ Analysis and Visualization Workshop at Supercomputing* in submission.

[11] Sage A Weil, Scott A Brandt, Ethan L Miller, Darrell DE Long, and Carlos Maltzahn. 2006. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th symposium on Operating systems design and implementation*. USENIX Association, 307–320.

[12] K Wu, S Ahern, E W Bethel, J Chen, H Childs, C Geddes, J Gu, H Hagen, B Hamann, J Lauret, J Meredith, P Messmer, E Otoo, A Poskanzer, O RÃijbel, A Shoshani, A Sim, K Stockinger, G Weber, W m Zhang, and et al. 2009. FastBit: Interactively Searching Massive Data. In *PROC. OF SCIDAC 2009*.

[13] Tzuhsien Wu, Hao Shyng, Jerry Chou, Bin Dong, and Kesheng Wu. 2016. Indexing blocks to reduce space and time requirements for searching large data files. In *Cluster, Cloud and Grid Computing (CCGrid), 2016 16th IEEE/ACM International Symposium on*. IEEE, 398–402.

[14] Xiaocheng Zou, David A. Boyuka, II, Dhara Desai, Daniel F. Martin, Suren Byna, and Kesheng Wu. 2016. AMR-aware in Situ Indexing and Scalable Querying. In *Proceedings of the 24th High Performance Computing Symposium (HPC '16)*. Society for Computer Simulation International, San Diego, CA, USA, Article 26, 9 pages. https://doi.org/10.22360/SpringSim.2016.HPC.012