

Using a Robust Metadata Management System to Accelerate Scientific Discovery at Extreme Scales

Margaret Lawson
Sandia National Laboratories &
Univ. of Illinois at Urbana-Champaign
mlawso@sandia.gov

Jay Lofstead
Sandia National Laboratories
Albuquerque, NM
gflfst@sandia.gov

Abstract—Our previous work, which can be referred to as EMPRESS 1.0, showed that rich metadata management provides a relatively low-overhead approach to facilitating insight from scale-up scientific applications [1]. However, this system did not provide the functionality needed for a viable production system or address whether such a system could scale. Therefore, we have extended our previous work to create EMPRESS 2.0, which incorporates the features required for a useful production system.

Through a discussion of EMPRESS 2.0, this paper explores how to incorporate rich query functionality, fault tolerance, and atomic operations into a scalable, storage system independent metadata management system that is easy to use. This paper demonstrates that such a system offers significant performance advantages over HDF5, providing metadata querying that is 150X to 650X faster, and can greatly accelerate post-processing. Finally, since the current implementation of EMPRESS 2.0 relies on an RDBMS, this paper demonstrates that an RDBMS is a viable technology for managing data-oriented metadata.

Index Terms—custom metadata, descriptive metadata, EMPRESS, SIRIUS, data tagging, Decaf, ATDM

I. INTRODUCTION

In recent years, supercomputers have become dramatically more computationally powerful. This has allowed scientists to run simulations at increasingly fine-grained spatial and temporal scales [2], [3]. As a result, an extended run of simulations such as S3D combustion [4], XGC edge plasma fusion [5], and GTS core plasma fusion [6] can easily generate datasets in the terabyte to petabyte range. While these fine-grained simulation outputs increase the opportunity for insight, the increasingly large datasets they produce are difficult to store, manage, and explore. This is compounded by the fact that memory capacity, network bandwidths, and disk bandwidths are growing at a much slower pace [7]. As a result, scientists can no longer rely on traditional methods of post-processing, which require loading the entire outputted dataset for analysis or visualization. Instead, scientists need an efficient way of identifying which portions of an outputted dataset contain features of interest and should be loaded for further analysis. Custom metadata offers a solution. Scientists can perform lightweight analysis to identify potential features of interest, store metadata about these features, and later use this metadata to determine which subsets of data are of interest and merit further analysis. This can greatly reduce the amount of data that is loaded. Since only a small amount of information is stored as metadata, it can be efficiently loaded and explored to

identify what data is of interest, thereby accelerating scientific discovery. Thus, a simplified version of the current paradigm is seen in Figure 1 while a simplified version of the custom metadata paradigm is seen in Figure 2.

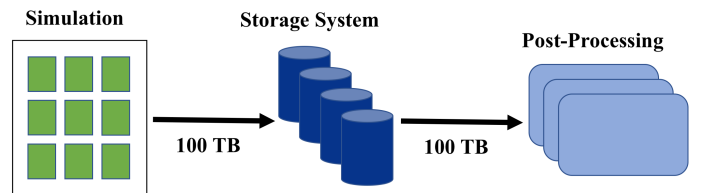


Fig. 1: Typical Write-Read Process (Simplified)

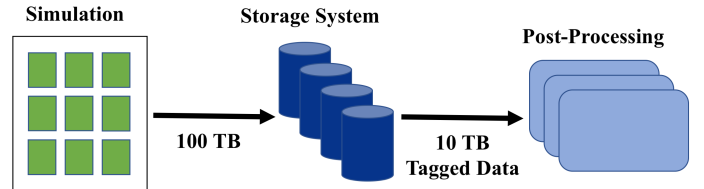


Fig. 2: Custom Metadata Write-Read Process (Simplified)

While EMPRESS 1.0 [1] provided evidence that rich metadata management could be used to accelerate scientific discovery, the system did not provide the functionality needed for a viable production system. Any viable solution for the search-and-discovery problem must include robust query functionality, fault tolerance, and atomic operations. To be truly useful, the system must also be portable, so that a user's data can move from one system to another without information loss, and easy to use. Finally, a viable solution must be scalable and introduce minimal overheads in terms of memory, latency and dedicated processor requirements. EMPRESS 2.0 provides these functionalities and with a .1% processor allocation overhead offers metadata writes that are faster than HDF5 at scales above 1000 compute cores, metadata reads that are 150X to 650X faster than HDF5, and approximately linear scalability. This paper offers the following contributions:

- **A Wide Range of Metadata Services.** EMPRESS 2.0 uses a conceptual metadata model that supports domain independent, extensible, user-defined metadata. This model enables EMPRESS to efficiently offer a wide range

of basic and custom metadata searches that support many kinds of analysis including global, spatial, temporal, and multivariate analysis.

- **Atomic Operations.** EMPRESS 2.0 offers users atomic operations through scalable transactions. These transactions allow clients to determine if and when to commit a transaction and thereby ensure the consistency and availability of metadata. This enables workflows to function safely and efficiently.
- **Fault Tolerance.** EMPRESS 2.0 provides scalable fault tolerance by allowing users to choose how to recover from failures occurring at the function, transaction, and hardware levels.
- **Portability.** EMPRESS 2.0 demonstrates that scalable efficiency can still be achieved with metadata that is portable to a wide variety of other systems (e.g., object systems and parallel file systems).
- **Easy to Use.** EMPRESS 2.0 demonstrates that it is possible to achieve high performance when using meaningful, high-level concepts to access metadata, even if users must first look up this high-level information.

In addition, this paper provides contributions through its evaluation of EMPRESS 2.0 by demonstrating the following:

- **Rich Metadata Can Accelerate Post-Processing.** While previous work has shown that rich metadata management can be offered with relatively good efficiency and has provided theoretical arguments that this can accelerate post processing [1], this claim has yet to be substantiated. This work is one of the first to evaluate the impact of rich, descriptive metadata management on post-processing and provides evidence that it can dramatically accelerate post-processing by efficiently identifying potential features of interest and limiting the reading scope to the associated data. We have not found any work that examines how a metadata management system of EMPRESS 2.0's breadth and depth can speed post-processing.
- **RDBMS is a Viable HPC Technology for Data-Oriented Metadata.** Anecdotally, RDBMSs are said to be unscalable and thus not a viable technology for managing descriptive (data-oriented) metadata for HPC applications. However, the implementation of EMPRESS 2.0 that was evaluated uses an RDBMS as the underlying infrastructure and, as mentioned above, provides substantially faster metadata management than HDF5. This provides evidence that RDBMSs can be a valuable, scalable technology for data-oriented metadata management for HPC. In particular, it suggests that an RDBMS can be used to create a hybrid system that provides both the well-tuned data access methods offered by databases and the low-storage overhead custom formats offered by traditional I/O libraries. While SciDB [8] has pioneered a database-hybrid approach for data management using a non-relational database, it has not extended this approach to descriptive metadata management and the robust set of queries this requires.

The rest of the paper is organized as follows. Section II contains an overview of the design elements. Section III discusses the implementation decisions. Section IV contains the testing and evaluation information. Section V presents related work, and Section VI discusses future work.

II. DESIGN

This section will provide a more in-depth exploration of how EMPRESS 2.0 efficiently offers the features needed for a production system.

A. Metadata Services

EMPRESS 1.0 only offered basic and custom metadata for variables and provided three kind of custom metadata searches: 1). list all metadata objects written by process X, 2). list all metadata objects of type Y written by process X and 3). list all metadata object types. EMPRESS 2.0 uses a vastly different metadata model that allows it to offer over 40 different efficient search functions.

1) *Metadata Model:* EMPRESS 2.0 uses a conceptual metadata model to support domain independent, extensible, user-defined metadata and to provide a wide range of metadata operations efficiently. This metadata model also makes EMPRESS 2.0 easy to use since it is based on runs, timesteps and variables, high-level constructs that application scientists are accustomed to. This model is displayed in Figure 3 and is composed of two broad categories: basic and custom metadata. Each basic metadata object can have an associated set of attributes, and each attribute is associated with a single tag (named label).

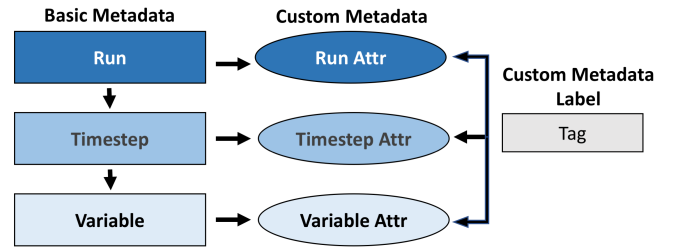


Fig. 3: EMPRESS 2.0's Metadata Model

a) *Basic Metadata:* Basic metadata captures the structure of a simulation output and simple metadata about the various components. Basic metadata is composed of three categories: (application) runs, timesteps (individual writes or data outputs), and variables (such as temperature or pressure). The basic run structure that EMPRESS 2.0 assumes is a hierarchical one. A simulation may be run one or more times, and each run is composed of timesteps. Each timestep is composed of variables. The variables and number of variables output may differ for each timestep. This basic metadata structure is domain-independent and allows users a great deal of flexibility.

b) Custom Metadata: Custom metadata refers to attributes (user-defined metadata objects) and their associated tags (user-defined labels for attributes). A tag indicates what kind of metadata is being stored. An attribute can be viewed as an instance of a particular tag that is associated with a basic metadata object. Attributes can be associated with an entire run, timestep, or a subset of a variable. Each attribute is associated with a user-defined tag and a value, which can be of any data type. Variable attributes also store the global spatial coordinates of the variable subset they refer to. For example, a scientist could add a “maximum” attribute with value 10K onto the “temperature” variable from $X:[75,80]$, $Y[60,80]$, $Z[300,325]$. Thus, EMPRESS 2.0 provides substantial flexibility and extensibility through its user-defined tags and attributes.

2) Metadata Queries: EMPRESS 2.0 allows users to retrieve basic metadata about the structure and conditions of application runs and custom metadata about features that match one or more conditions. These conditions include the type of feature (label), its spatial or temporal location, its associated variable or variables, and its associated value, where the value must be below, between or above the given value(s). Through these functionalities, EMPRESS 2.0 facilitates global, spatial, temporal and multivariate analysis, none of which EMPRESS 1.0 facilitated.

a) Global Analysis: EMPRESS 2.0 allows users to store and query global metadata through use of run attributes. Users can then perform global analysis such as identifying trends across sets of application runs, which simulation runs are of interest and merit further analysis, and the simulation conditions that produce a phenomenon of interest. Users can, for example, flag a run as containing a combustion event and later query which runs contain such an event. Users could also store the maximum temperature achieved during the course of the run and later query for the list of runs that exceed a certain maximum temperature. Run attributes can also be used for storing notes containing information about the run such as the units used, a note about the conditions, or a note containing high-level information about what the researcher observed. Users can also retrieve basic global metadata such as the names of stored simulation runs or the number of timesteps for a specific run. These kinds of global analysis are useful in many different scenarios and are especially critical for ensemble simulations.

b) Spatial Analysis: EMPRESS 2.0 allows users to store spatial features and perform location-dependent queries. This is important since spatial metadata allows users to determine if an observed phenomenon is of interest for the given analysis. For example, in a plasma fusion reactor simulation, an area of turbulence might only be of interest if it is a certain size or is located near the reactor edge or in a weather simulation, a storm cell might only be of interest if it makes landfall. Thus, EMPRESS 2.0 allows users to restrict queries to a particular spatial area, or to retrieve a list of timesteps where a feature appears in a given spatial area.

c) Temporal Analysis: EMPRESS 2.0 also facilitates temporal analysis. EMPRESS 2.0 allows users to identify when a feature first develops and to track how the feature progresses over time. A user can, for example, retrieve a list of timesteps where a storm cell appears and retrieve the spatial and value information for the storm cell. Users can also use timestep attributes to record high-level information about an entire timestep such as the highest temperature achieved across the simulation space at that time. These “maximum temperature” timestep attributes could then be used to perform rapid time series analysis. EMPRESS 2.0 also allows users to retrieve additional information about a particular timestep. For a given timestep, a user can request a catalog of the kinds of features that appear or that appear in a given space. This kind of analysis is useful for determining if a timestep is in fact of interest, or for sampling approaches. EMPRESS 2.0 also facilitates more basic operations such as determining at which timesteps a particular variable is output. This is crucial since many simulations do not output all variables at each timestep.

d) Multivariate Analysis: Finally, EMPRESS 2.0 aids with multivariate analysis, thereby allowing users to track complex trends and interactions. EMPRESS 2.0 allows users to store multivariate features through use of run and timestep attributes. Users can, for example, perform a calculation using multiple variables and store the result as a timestep or run attribute. Users can additionally store a feature as a variable attribute and restrict their queries to one or multiple variables. All variable attributes can be searched using either a particular variable or a variable name substring. This allows multiple variables to be queried at once.

B. Atomic Operations

While EMPRESS 1.0 provided atomic transactions, this system was not optimized or formally evaluated. EMPRESS 2.0 builds on this system to offer users atomic operations through scalable transactions. EMPRESS 2.0 provides transaction management services that are largely based on the D²T system [9]. However, unlike the D²T system, it does not require clients to confirm how many will be participating in a given transaction and it does not use sub-transactions. Transactions are managed as follows. For every piece of metadata that is written, the user provides a transaction id. Users can then commit or “activate” all metadata associated with this id, using a set of EMPRESS 2.0 functions (one per metadata type, e.g., run, timestep, variable). Metadata that has not yet been activated can be queried only by supplying the matching transaction id. All other queries will ignore inactive metadata. This allows users to delay committing their metadata until they have ensured its correctness and until the associated data is available for reading. In short, EMPRESS 2.0 allow clients to determine what operations should be considered a transaction and when to commit a transaction and make it visible to other processes. This eliminates a need for server coordination. This transaction management ensures the consistency and availability of metadata and therefore ensures that workflows can work safely and efficiently.

C. Fault Tolerance

While EMPRESS 1.0 did not provide fault tolerance, EMPRESS 2.0 provides scalable fault tolerance by allowing users to choose how to recover from failures occurring at the function, transaction, and hardware levels. EMPRESS 2.0 provides return values from all EMPRESS functions, allowing users to handle operation failures as they so choose. As mentioned above, EMPRESS 2.0 also provides fault-tolerance at the transaction level by allowing users to determine if and when to commit a transaction and make it visible to other operations. EMPRESS 2.0 also allows users to later delete pieces of metadata, in case an error is discovered. EMPRESS 2.0 also provides policies that deal with failures of the metadata managers. The stored metadata is periodically check-pointed to disk to prevent metadata loss, and the user can adjust this frequency to meet their needs. A similar approach is used by other scale-out data stores such as Cassandra [10] and PostgreSQL [11]. As will be demonstrated in Section IV, checkpointing the databases to disk can be done relatively quickly. But, more importantly, in most cases this checkpointing will not affect execution time. In general, applications alternate between computation and writing, and the metadata servers will be left idle during the computation phases. Therefore, if users checkpoint the databases at the end of a writing phase, the metadata servers will have the entire next compute phase to write out the databases. The relatively small size of metadata means it can be easily written during this time. The exception is that the final checkpoint (at the end of writing) will affect performance since the clients will have finished all computations. Finally, if an operation takes longer than a certain period of time, the client can contact EMPRESS 2.0 to determine if the metadata manager is still available and if not, connect to a new one. Thus, through a set of opt-in and user adjustable fault tolerance policies, EMPRESS 2.0 provides users with scalable fault tolerance that can be adjusted to meet their needs.

D. Portability

Once users have identified a potential feature of interest, they need an efficient way to determine what data is associated with that metadata. To provide this efficiently, most metadata management systems, such as SoMeta [12], directly store the address or named location of the associated data. However, this results in increased memory requirements and inhibits portability of the metadata, making it unusable if the storage location or storage system changes (e.g., from object system to parallel file system). Since scientific data is often moved between storage tiers or shared across storage systems, it is vital that the metadata be easily portable. While EMPRESS 1.0 provided portable metadata, it did not provide an efficient way to map from metadata to the associated data. EMPRESS 2.0 expands on this to offer both portable metadata and efficient mapping from metadata to the associated data. EMPRESS 2.0 offers this portability since it stores only logical, spatial coordinates from the simulation space and not physical data

locations. EMPRESS 2.0 then uses a novel object name generator that uses $O(1)$ storage space, to map user-meaningful metadata and logical spatial dimensions to the set of matching object or file names. This provides highly efficient object name generation both for writing and reading, and makes the metadata storage-system independent. A full discussion of the mapping function is outside the scope of this paper but can be found in a forthcoming paper being led by Reza Nasirigerdeh.

E. Easy to Use

Any system should allow a user to work in terms they are accustomed to rather than relying on internal system shortcuts. While arbitrary IDs may be “efficient” for systems implementors, these identifiers are not meaningful and must be tracked. While EMPRESS 1.0 required users to remember random identifiers, EMPRESS 2.0 does not, and instead relies on high-level concepts that application scientists are accustomed to such as runs, timesteps and variables. EMPRESS 2.0 also does not require users to remember even basic identifiers, such as the name of a simulation, the number of timesteps used in a simulation or the names of variables or tags. Since scientists often run a simulation many times and utilize various simulations, even this basic information may be forgotten. For this reason, EMPRESS 2.0 provides functions that catalog basic metadata about stored runs, timesteps, and variables. EMPRESS 2.0 also provides catalog functions for run, timestep and variable attributes so that if users are not sure what kind of metadata they are looking for, they can list all of the possibilities. Because of these highly efficient services, users are not required to remember identifiers and are not penalized for forgetting. No other metadata service offers this kind of catalog functionality. Finally, EMPRESS 2.0’s metadata model allows users to employ the same tag name across variables, timesteps, or runs. Users therefore do not have to worry about remembering which unique names have already used. This stands in contrast to the many metadata services that use key-value pairs and require users to remember which metadata object names they have used. Thus, EMPRESS 2.0 demonstrates that it is possible to achieve high performance by using high-level concepts to access metadata and by providing users with a means to look up this high-level information if they have forgotten it.

III. IMPLEMENTATION

Although EMPRESS 1.0 and EMPRESS 2.0 are very different systems, they have similar implementations. The current implementation of EMPRESS 2.0 and the decisions for this implementation are discussed below.

A. Client-Server Model

EMPRESS 2.0 uses in-memory RDBMSs for metadata storage and a set of independent, distributed, share-nothing servers. This client-server model is depicted in Figure 4 and will be discussed in detail below.

a) *RDBMS*: Each server maintains an in-memory RDBMS. RDBMSs, which use B-tree variants both for storage and indexes, have a time complexity of $\Theta(\log n)$ for basic operations such as write, read, and update whereas most hashing techniques offer $O(1)$ expected performance. However, hash tables cannot efficiently offer the wide range of queries we support such as variable attribute queries with any combination of spatial, temporal, variable, tag and value constraints. There is no way to optimize a single hash table key for all of these combinations. These inefficiencies are compounded by the fact that keys and values are often both stored as strings, requiring extensive string matching to determine which keys and values match a given query. An RDBMS can, by contrast, store metadata in its original datatype resulting in more efficient searches.

b) *Independent*: EMPRESS 2.0 uses a set of dedicated server processes running in parallel. The following is the query process: the client makes calls using the EMPRESS 2.0 API library, the EMPRESS 2.0 client sends a message to the server's queue, the server interacts with the database and returns the result to the EMPRESS 2.0 client which passes it to the client. Using dedicated servers removes the need for client-side coordination for write, storage, and read but requires the allocation of nodes that could otherwise be used for computation and introduces a new potential bottleneck. EMPRESS 2.0 tries to mitigate these concerns by offering efficient services with minimal dedicated resources (see Section IV) and allowing servers to be dynamically allocated to respond to server shortages or surpluses.

c) *Distributed, Shared-Nothing*: Each server maintains a copy of all basic metadata and a horizontal shard of all user-defined attributes, allowing distributed query processing. When EMPRESS 2.0 is initialized, each client is assigned to a single server by a method that ensures the number of clients per server is balanced. The shared-nothing design eliminates costly server-side coordination and offer clients flexibility in how they distribute metadata across the servers and perform read queries.

B. Using EMPRESS 2.0

EMPRESS 2.0 is a fully functioning, open-source system that can be found at <https://github.com/mlawsonca/empress>. EMPRESS 2.0's functionality is exposed to the user as a C++ library. While most of the metadata search functionalities discussed above can be executed with a single EMPRESS 2.0 function, two might require more than one function call and deserve clarification. First, in order to search across multiple runs for a potential feature of interest, the feature must either be stored as a run attribute or the user must catalog the list of runs and for each run perform a search for the feature. While EMPRESS 2.0 could provide a single function that searches multiple runs for a variable attribute of interest, this has not been implemented at this time. Second, EMPRESS 2.0 does not provide explicit support for multivariate queries. Instead, as mentioned above, the functionality is provided through run and timestep attributes or by having users store a multivariate metadata object as a single variable attribute and, if necessary, restricting the search to variable names matching a common substring. In both of these cases, users may perform the search using a single function call if they use run and/or timestep attributes, but these attributes may not be the most natural fit since they do not provide direct support for spatial queries. It also should be mentioned that EMPRESS 2.0 does not require users to checkpoint the stored metadata to disk. Instead, it provides this check-pointing as a function, and users can determine how frequently they wish to use it. The current function assumes that each server will write its database to a file. More general, flexible alternatives will be explored in the future. An additional area of future work is finding a more efficient checkpointing system. The current system maintains the metadata in memory even after it has been checkpointed, and then during checkpointing copies the entire database to disk. Therefore, the same metadata is repeatedly checkpointed to disk, and checkpoints take increasing amounts of time as the database size increases. Nevertheless, as discussed above (see Section II-C), we do not expect the checkpointing system to alter performance since checkpointing can be done in the background during compute phases. If, however, there is a situation under which this checkpointing time affects performance, we would recommend using the following estimator developed by Daly [13] to determine the optimum time interval between checkpoints: $T_{opt} = \sqrt{2C(M + R)} - C$ where T_{opt} is the optimum time interval between checkpoints, C is the time to write a checkpoint, M is the mean time between system failures, and R is the restart overhead. Since, with our current

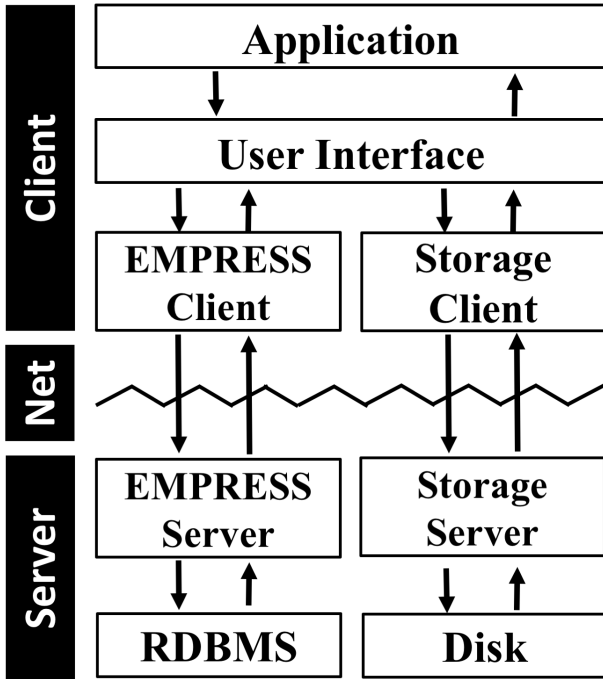


Fig. 4: Client-Server Model

method, the cost per checkpoint will increase approximately linearly with the size of the database, we would recommend using the estimated median checkpointing cost (which could be extrapolated using our checkpoint time vs. database size results) or using the maximum estimated checkpointing cost as an upper bound.

C. External Libraries

EMPRESS 2.0 uses SQLite [14] [15] version 3.22 as the RDBMS. SQLite was chosen because of its server-less model, dynamic type system, and light-weight design. EMPRESS 2.0 uses Faodel [16] for message passing between the clients and servers. Faodel is built upon the long stable and performant NNTI RDMA communication layer from the Nessie [17] RPC library from Sandia and provides asynchronous message passing and message queuing. EMPRESS 2.0 uses the Boost serialization library to serialize the data passed as messages between the client and servers and to store non-native types in SQLite.

IV. EVALUATION

To evaluate if EMPRESS 2.0 can provide a useful production tool for extreme-scale scientific simulations, we evaluate its scalability, its storage, write and read overheads, its fault-tolerance and transaction performance, its comparison to alternatives and its ability to accelerate data exploration.

A. Testing Configurations

Testing is performed on the Skybridge capacity cluster at Sandia, which has 1848 nodes with 16 cores/node and uses 2.6 Ghz Intel Sandy Bridge processors. It has an Infiniband interconnect, 4 GB RAM per process, and runs RHEL7. We use the GNU C++ compiler version 4.9.2, OpenMPI 1.10, and HDF5 1.10 (choice discussed below). All experiments utilize the Lustre parallel file system, HDF5’s default file system parameters, and HDF5 chunking. Tests are also run on the Chama and Serrano capacity clusters at Sandia but show similar results and are omitted for space considerations. The tests use all 16 cores per node (where possible).

To evaluate EMPRESS 2.0, we run two series of tests with the same setup. One uses HDF5 [18] for metadata management while the other uses EMPRESS 2.0. Both use HDF5 for data management. HDF5 was chosen since it is the most frequently used I/O library for HPC science applications [19] and thus presents a realistic representation of the metadata management used by scientists today. In addition, HDF5 offers superior metadata management to alternatives since it offers scoped attributes and user-defined datatypes for attributes. HDF5 also supports tables with variable length entries (packet tables), which provide a compact representation of attributes that can be iterated over easily [18]. We use packet tables to extend HDF5 to offer EMPRESS 2.0’s functionality.

Tests are performed using 1000, 2000, and 4000 processes for writing, and one tenth as many processes for reading (100, 200, 400). All EMPRESS 2.0 tests use an 1000:1 client-server ratio for writing and a 100:1 client-server ratio for reading.

TABLE I: Testing Configurations

Test Type	Number of Write Processes	Number of Metadata Servers
EMPRESS 2.0 + HDF5	1000	1
EMPRESS 2.0 + HDF5	2000	2
EMPRESS 2.0 + HDF5	4000	4
HDF5	1000	N/A
HDF5	2000	N/A
HDF5	4000	N/A

These configurations are summarized in Table I. Each of these configurations is performed a minimum of five times, and results are averaged across these runs. The choice of these configurations merits some discussion. First, the 1000:1 client-server ratio for writing is chosen to simulate the expected use case: that scientists will wish to allocate as few hardware resources as possible to metadata management since they could otherwise be used to perform additional computations. One tenth as many client processes are used for reading (vs. writing) since, in general, scientists will allocate far more resources to computation than they will to post-processing. Finally, the 100:1 client-server ratio is chosen for reading because, with 100, 200, and 400 read clients, 100:1 is the largest fixed ratio that could be used for all configurations (thereby allowing us to evaluate the system’s weak scaling).

It is important to note that EMPRESS 2.0 can be used with a wide range of allocated resources. Additional testing that is outside of the scope of this paper evaluated how EMPRESS 2.0 performs with 10X as many resources for writing (an 100:1 client-server ratio). These results indicate that allocating 10X more EMPRESS 2.0 resources produces a 8X-9X speedup in metadata writing. This difference between resource allocation and performance is partially due to the fact that the basic metadata (about the application run, timesteps, variables, and tags) must be written to each server. Also, with the additional servers, there are fewer clients per server and thus a greater chance of load imbalance. Although clients are equally distributed across the servers, it is randomly determined which clients will write an attribute for a given variable and tag (subject to the tags frequency). Therefore, we would expect this difference to become even more pronounced with even lower client-server ratios. We have not yet evaluated EMPRESS 2.0 with even fewer dedicated resources. It is likely that at a much higher client-server ratio, EMPRESS 2.0’s performance will degrade significantly since the large number of incoming messages per server could simulate a denial-of-service attack.

B. Writing

Each test writes a single application run with 3 timesteps. Each timestep is composed of a set of 10 3D variables. Variables used in this evaluation include temperature, pressure, and density among others. Each of these variables is distributed across the processes using a 3D domain decomposition, so that each process writes a regular hyper-rectangle (a “chunk”)

TABLE II: Tags used in Testing

Tag	Data Type	Frequency (% of all chunks)
Frequent Blob	Boolean	25%
Infrequent Blob	Boolean	5%
Rare Blob	Boolean	.1%
Maximum	Double	100%
Minimum	Double	100%
Frequent Note	String	20%
Infrequent Note	String	2.5%
Rare Note	String	.5%
Infrequent Range	2 Integers	10%
Rare Range	2 Integers	1%

per variable that is .04 GB large. This size is chosen so that each process writes 10% of its total RAM per timestep, which is a typical amount. 10 tags are used, each of which has a set frequency that determines what percentage of chunks it is associated with. These tags include “blobs” (a scientific name for spatial phenomena), annotations, ranges, and maximum and minimum. The blobs have a Boolean value (indicating presence or absence of a particular feature), the maximum and minimum have a double value (like the associated data), the notes have text values, and the ranges have values that are a pair of integers. These tags are summarized in the Table II. On average, 2.6 attributes are written per chunk (per variable, per timestep). For each timestep, the global maximum and minimum for the “temperature” variable are written as timestep attributes and the maximum and minimum across all timesteps are inserted as run attributes.

C. Reading

Reading consists of 2 stages. The first stage performs six read patterns that have been identified as typical for analysis codes [20]. These six patterns are, for a given timestep: read all data, read all data for a variable, read all data for 3 variables, read a plane in each dimension, read a 3D subspace, and read a partial plane in each dimension. The second stage examines how rich metadata can be used to accelerate these read patterns. Patterns 2 and 3 (read a single variable and read three variables) are performed using three different read selectivities. This process involves querying a particular feature of interest (tag) that appears with the given selectivity (found on 25%, 5% and .1% of the data chunks), and then reading in only the data that matches this query.

D. Write and Read Examples

These are the write and read processes used in the testing harness.

a) Writing: Algorithm 1 demonstrates the basic write process for an application run. At the start of writing, each compute process initializes the EMPRESS 2.0 client (using `metadata_init`) which then connects to a single EMPRESS 2.0 server. The functions that start with * are performed by a single client. The functions start with ** are called by one client per server. The remaining functions are performed by each

client. This ensures that each server is given a copy the basic metadata (runs, timesteps, variables) and tags, and that the variable attributes are distributed across the servers. A single client is in charge of writing the timestep and run attributes. The “batch” functions indicate that all of the metadata writes of the same type have been batched (sent as a single message). All testing runs use this “batch” insertion since initial tests revealed that at the evaluated scales, the individual metadata writes take five to ten times longer than the batched writes due to the large number of small messages. The client can decide for each of these types of metadata when and if to commit them.

Algorithm 1 Metadata write algorithm

```

1: procedure WRITERUN
2:   metadata_init (...)
3:   **metadata_create_run (...)
4:   **metadata_create_tags_batch (...)
5:   for all timesteps do
6:     **metadata_create_timestep (...)
7:     **metadata_create_vars_batch (...)
8:     for all variables do
9:       write_chunk_data (...)
10:    end for
11:    *metadata_insert_timestep_attributes_batch (...)
12:    metadata_insert_var_attributes_batch (...)
13:  end for
14:  *metadata_insert_run_attributes_batch (...)
15: end procedure

```

b) Reading: An example metadata exploration and read procedure is presented in Algorithm 2. Again, the first step for reading is to initialize the EMPRESS 2.0 client. Then, the set of stored runs are cataloged to determine which to explore, and then, for this run, the stored timesteps are cataloged. The user then catalogs the variables that were output for a timestep of interest and catalogs the tags that are associated with the run. This is then used to catalog all attributes associated with this variable, a tag of interest, and a logical spatial location (such as “temperature”, “maximum”, $X:[50,100]$, $Y:[100,200]$, $Z:[100,200]$). Each process then iterates over the list of attributes and reads in the data that overlaps with its assigned portion of the simulation space. In practice we find that client-side coordination is more efficient than having all clients send the same queries individually, so basic metadata is cataloged by a single client, serialized and broadcast (denoted by *) and attributes are cataloged once per server, serialized, and allgathered (denoted by **).

c) HDF5 Writing and Reading: The HDF5 metadata write and read processes have a few differences. Since the attribute tables are stored internally, processes must open the associated timestep file, open the associated dataset (for variable attributes), and then open the attribute table. We follow HDF5’s advice for how to orchestrate metadata writes and reads. A single process opens the file individually, writes or reads all relevant metadata, and closes the file. All processes

Algorithm 2 Metadata read algorithm

```

1: procedure READVARATTRS
2:   metadata_init (...)
3:   *metadata_catalog_runs (...)
4:   *metadata_catalog_timesteps (...)      ▷ For Run X
5:   *metadata_catalog_vars (...)           ▷ For Timestep Y
6:   *metadata_catalog_tags (...)           ▷ For Run X
7:   **md_catalog_all_var_attrs_with_tag_var_dims (...)
8:   for all attrs do
9:     if dims_overlap(proc_dims, attr_dims) then
10:      read_attr(attr, overlapping_dims)
11:    end if
12:  end for
13: end procedure

```

TABLE III: EMPRESS 2.0 Storage Overheads

# Write Procs	Data Size	Metadata Size	EMPR Storage Overhead
1000	1.2 TB	8.1 MB	.0007%
2000	2.4 TB	16.2 MB	.0007%
4000	4.8 TB	32.6 MB	.0007%

then open the file collectively to write or read the data in parallel. This minimizes the overhead since all metadata operations on a collectively opened file must be performed collectively.

E. Results

a) Storage Overhead: Table III demonstrates EMPRESS 2.0's storage overhead. The results show that EMPRESS 2.0 can support basic and rich metadata and can provide mapping from logical spatial dimensions to physical storage locations with a trivial amount of required storage space. The relatively small size of the metadata demonstrates why there are significant advantages to loading and exploring the metadata (as opposed to the data) to identify features of interest.

b) Write and Read Overheads: Tables IV and V demonstrate that EMPRESS 2.0 introduces a very small write and read penalty. Even with minimal resource requirements of an 1000:1 client-server ratio for writing and 100:1 for reading, EMPRESS 2.0's metadata writes and reads constitute a small fraction of data write and read times.

c) Database Checkpointing: At the end of writing, each server outputs its database to disk. Figure 5 demonstrates the performance per server for this checkpointing with medians represented by the orange lines, 25th and 75th percentiles demonstrated using the boxes and the maxima and minima indicated by the whiskers (the header and footer lines). As expected, the performance is relatively constant across the number of clients since the number of clients per server (and thus the database size per server) is held constant. These results indicate that the database can be checkpointed to disk relatively quickly, and thus that EMPRESS 2.0 offers a low-cost fault-tolerant system.

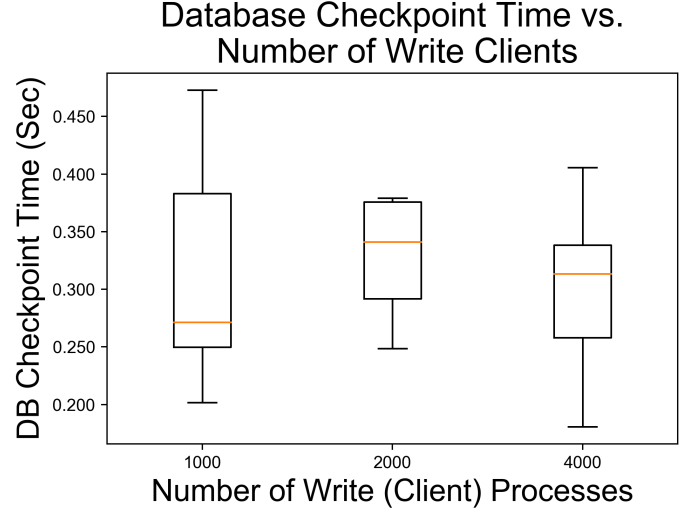


Fig. 5

d) Transactions: As discussed above (see Section II-B), EMPRESS 2.0 has a set of functions to commit metadata write transactions to the database, thereby making the metadata visible to all queries. The average performance for these commit operations is shown in Figure 6. This performance is largely determined by two factors: the number of metadata objects being activated and the total number of metadata objects of the type being activated. EMPRESS 2.0 maintains an index on the transaction id for each piece of metadata, allowing it to find the metadata objects to activate in $O(\log(n))$ time. It then updates each of these metadata objects to be "active." The performance results demonstrate that this system of transaction management is a low-cost way to ensure the consistency and availability of data.

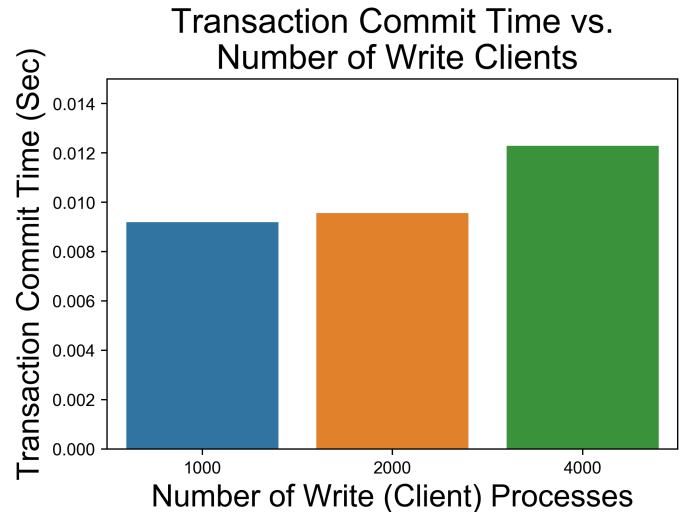


Fig. 6

e) Write Comparison: Table IV demonstrates how EMPRESS 2.0's performance compares to HDF5 for metadata writes. This information is also presented visually in Figure 7. At small scales, HDF5 actually out-performs EMPRESS 2.0 since it has a single client gather and write the metadata

TABLE IV: Metadata Write Overheads

# Write Procs	Data Write	EMPR Md Write	EMPR Md Overhead	HDF5 Md Write	HDF5 Md Overhead
1000	1753s	1.53s	0.09%	0.66s	0.04%
2000	3852s	1.65s	0.04%	1.80s	0.05%
4000	7406s	1.56s	0.02%	3.22s	0.04%

TABLE V: Metadata Read Overheads for Pattern 2

# Read Procs	Data Read	EMPR Md Read	EMPR Md Overhead	HDF5 Md Read	HDF5 Md Overhead
100	12.5s	.03s	0.2%	11.7s	93.6%
200	38.7s	.14s	0.4%	23.2s	59.9%
400	88.5s	.07s	0.1%	48.0s	54.2%

while EMPRESS 2.0 clients individually perform thousands of smaller writes. Clients using EMPRESS 2.0 could mitigate this discrepancy by performing similar write aggregations. The results show that as the number of processes increases, the HDF5 metadata write performance degrades while the EMPRESS 2.0 performance remains constant if the client-server ratio is held constant. HDF5 suffers from the fact that it has no means of scaling out and parallelizing its metadata operations.

Metadata Write Time: EMPRESS vs. HDF5

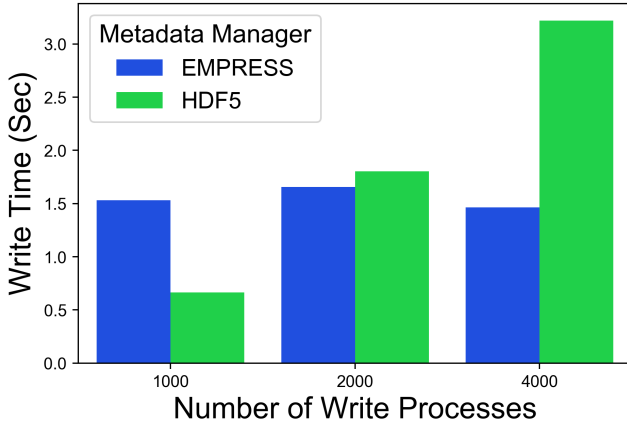


Fig. 7

f) *Read Comparison:* Tables V demonstrates the summed metadata query time that each system uses to perform the three metadata queries for pattern 2 (retrieving all attributes for a given variable that match a tag of 25%, 5% and .1% selectivity). This information is also presented visually in Figure 8. The results demonstrate that while EMPRESS 2.0 can perform the queries very efficiently, HDF5 cannot since it has no means of scaling out or parallelizing metadata querying and does not offer indexes (and must therefore perform a linear search). At the moderate scales evaluated, HDF5 uses almost as much time to perform the metadata query as to read the data, which presents an unacceptably large performance penalty. These findings confirm that the metadata support offered by current I/O tools are not sufficient.

g) *Accelerating Data Exploration:* Figure 9 compares the time to read an entire variable with the time for EMPRESS

Metadata Read Time for Pattern 2: EMPRESS vs. HDF5

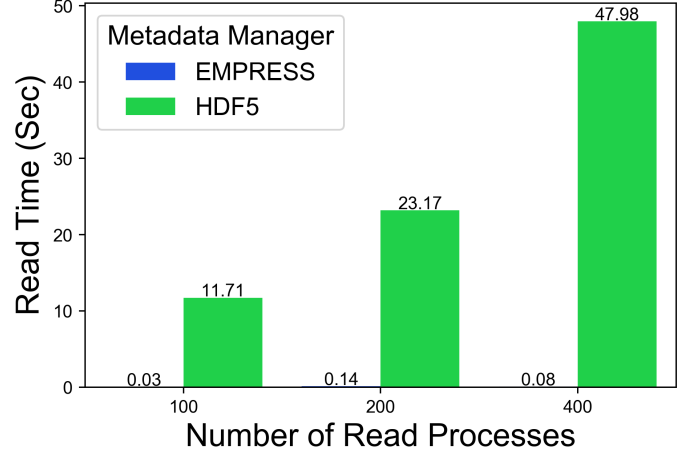


Fig. 8

2.0 to identify tagged areas and to read the associated data (at various selectivities). The results demonstrate that even with the small performance overhead introduced by the metadata operations, EMPRESS 2.0 can lead to significant gains by reducing the volume of data read. Since I/O is a rate limiting factor, EMPRESS 2.0 can thus greatly accelerate scientific exploration and thus discovery. However, it is important to note that reducing the read volume does not produce a 1-to-1 reduction in the total read time. This finding is due largely to the fact that HDF5 has to linearly search its stored metadata to identify the matching chunk and that attributes are distributed randomly, resulting in the possibility of storage system contention. Nevertheless, the results demonstrate that, as we push towards the exascale age, metadata management represents a critical tool for scientific discovery, and an RDBMS-based approach offers significant advantages.

Single Variable Read Time Vs. Selectivity

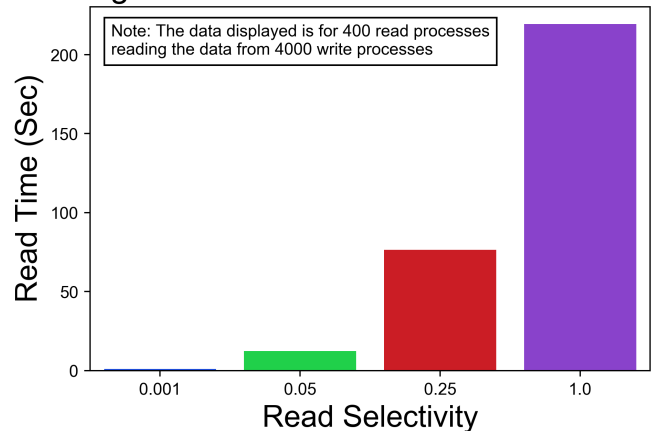


Fig. 9

V. RELATED WORK

a) *Storage Systems:* Most storage systems are largely focused on offering scalable storage and fault-tolerance and do not offer “tagging” and “searching” capabilities [12]. One

exception is HP StoreAll with ExpressQuery [21], a production archival storage system that supports file tags and tag searches but suffers a number of limitations. Tags are stored as string key-value pairs, requiring extensive string matching to perform queries. The system also requires users to generate and remember unique names. Tags are also associated with entire files, which does not offer sufficiently fine-grained indexing for simulation outputs, which tend to put an entire variable or even entire timestep in a file.

b) Metadata Management and at the Storage Level:

Low-level metadata management systems, such as TagIt [22], can increase consistency but incur several costs. First, since scientific users generally use I/O systems, they cannot leverage the metadata system unless the I/O system is extended to interface with it. Second, these systems tend to be dependent on the specific storage back-end, limiting portability. Third, many file systems already experience a severe metadata bottleneck, and have no means of dynamically increasing the number of metadata servers to meet demand. Fourth, compute nodes offer greater processing power and faster interconnects than storage nodes. A few TagIt specific problems are that it does not support sub-file indexing, is reliant on the file system's extended attributes, and supports very few search queries.

c) I/O Systems: The four most common I/O systems used by scientific simulations are ADIOS [23], PnetCDF [3], HDF5 [18], and netCDF-4 [24]. While each of these offer user-defined metadata attributes, they do not support scalable metadata services such as tagging, searching, or indexing. Furthermore, for each of these, metadata is embedded in associated data files, which makes it difficult to efficiently offer global, full-context metadata views. In addition, each of these only uses a single client to write and read all metadata, adding a severe scalability bottleneck.

d) Metadata Management at the I/O Level: SoMeta [12] offers many similar capabilities to EMPRESS 2.0, such as "tag and search." However, while EMPRESS 2.0 uses an RDBMS backend with indexes to speed querying, SoMeta uses a distributed hash table backend (without indexes). This makes SoMeta subject to many of the limitations described above (see Section III-A). In addition, SoMeta requires users to generate and remember unique metadata object identifiers and does not support transactions, limiting its use in workflows.

Several related tools have been developed for a specific domain or application such as the Catalog Archive Server (CAS) [25] for the Sloan Digital Sky Survey (SDSS), ATLAS [26] for the Large Hadron Collider, and the Atmospheric Data Discovery System (ADDS) [27]. These systems offer an extensive range of metadata queries, but are entirely domain or simulation specific and do not support user-defined attributes or tags.

e) SIRIUS: EMPRESS 2.0 is part of a larger work, the DOE Office of Science ASCR SIRIUS project. SIRIUS is designed to provide a system for scientific data management at exascale. Specific contributions of SIRIUS include predictable storage performance (provided by Ceph [28]), extensible custom metadata (provided by EMPRESS 2.0), high compression

with small error bounds, and different level-of-precision read options, speeding data access for lower precision reads.

VI. FUTURE WORK

Future work will involve more extensive evaluation of EMPRESS. Future evaluation will examine how EMPRESS's performance varies with different amounts of metadata and at much higher scales. We will also compare EMPRESS to other dedicated metadata management systems such as [12] SoMeta. In addition, we will investigate implementation decisions such as when it is advantageous to use a NoSQL database instead of an RDBMS and whether a different RDBMS would better serve our goals.

Future work will also focus on expanding EMPRESS's functionality. We will investigate the possibility of offering direct support for coordinate systems other than Cartesian and for supporting non-uniform meshes and Adaptive Mesh Refinement codes. It will also be important to explore how a metadata system like EMPRESS can better serve applications with different data models, such as genomics applications. Although EMPRESS 2.0's metadata model can be applied to any data model that can be viewed hierarchically (with three or fewer levels), its model is not the most natural fit for some applications. Other integration models, such as a metadata management system designed to facilitate in situ or in transit analysis, should also be investigated. Although EMPRESS can respond simultaneously to both write and read queries, it is not optimized for accelerating in situ analysis. Finally, we hope to explore more fully how a metadata management service like EMPRESS 2.0 can be integrated with storage systems to make better decisions about prefetching, tiering, and striping.

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] M. Lawson, C. Ulmer, S. Mukherjee, G. Templet, J. Lofstead, S. Levy, P. Widener, and T. Kordenbrock, "Empress: Extensible metadata provider for extreme-scale scientific simulations," in *Proceedings of the 2nd Joint International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems*, ser. PDSW-DISC '17. New York, NY, USA: ACM, 2017, pp. 19–24. [Online]. Available: <http://doi.acm.org/10.1145/3149393.3149403>
- [2] Z. Gong, S. Lakshminarasimhan, J. Jenkins, H. Kolla, S. Ethier, J. Chen, R. Ross, S. Klasky, and N. F. Samatova, "Multi-level layout optimization for efficient spatio-temporal queries on isabela-compressed data," in *Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*. IEEE, 2012, pp. 873–884.

- [3] J. Li, W. keng Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, and M. Zingale, "Parallel netcdf: A high-performance scientific i/o interface," in *Supercomputing, 2003 ACM/IEEE Conference*, Nov 2003, pp. 39–39.
- [4] J. H. Chen, A. Choudhary, B. De Supinski, M. DeVries, E. R. Hawkes, S. Klasky, W.-K. Liao, K.-L. Ma, J. Mellor-Crummey, N. Podhorszki *et al.*, "Terascale direct numerical simulations of turbulent combustion using s3d," *Computational Science & Discovery*, vol. 2, no. 1, p. 015001, 2009.
- [5] S. Ku, C. Chang, and P. Diamond, "Full-f gyrokinetic particle simulation of centrally heated global itg turbulence from magnetic axis to edge pedestal top in a realistic tokamak geometry," *Nuclear Fusion*, vol. 49, no. 11, p. 115021, 2009.
- [6] W. Wang, Z. Lin, W. Tang, W. Lee, S. Ethier, J. Lewandowski, G. Rewoldt, T. Hahm, and J. Manickam, "Gyro-kinetic simulation of global turbulent transport properties in tokamak experiments," *Physics of Plasmas*, vol. 13, no. 9, p. 092505, 2006.
- [7] Y. Wang, Y. Su, and G. Agrawal, "Supporting a light-weight data management layer over hdf5," in *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*. IEEE, 2013, pp. 335–342.
- [8] P. G. Brown, "Overview of scidb: large scale array storage, processing and analysis," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 2010, pp. 963–968.
- [9] J. Lofstead, J. Dayal, K. Schwan, and R. Oldfield, "D2t: Doubly distributed transactions for high performance and distributed computing," in *Cluster Computing (CLUSTER), 2012 IEEE International Conference on*. IEEE, 2012, pp. 90–98.
- [10] A. Lakshman and P. Malik, "Cassandra: structured storage system on a p2p network," in *Proceedings of the 28th ACM symposium on Principles of distributed computing*. ACM, 2009, pp. 5–5.
- [11] [Online]. Available: <http://www.postgresql.org/>
- [12] H. Tang, S. Byna, B. Dong, J. Liu, and Q. Koziol, "Someta: Scalable object-centric metadata management for high performance computing," in *Cluster Computing (CLUSTER), 2017 IEEE International Conference on*. IEEE, 2017, pp. 359–369.
- [13] J. T. Daly, "A higher order estimate of the optimum checkpoint interval for restart dumps," *Future generation computer systems*, vol. 22, no. 3, pp. 303–312, 2006.
- [14] [Online]. Available: <http://www.sqlite.org/>
- [15] M. Owens, "Embedding an sql database with sqlite," *Linux Journal*, vol. 2003, no. 110, p. 2, 2003.
- [16] C. Ulmer, S. Mukherjee, G. Templet, S. Levy, J. Lofstead, P. Widener, T. Kordenbrock, and M. Lawson, "Faodel: Data management for next-generation application workflows," in *Proceedings of Workshop on Infrastructure for Workflows and Application Composition (IWAC), 2018*, 2018.
- [17] R. A. Oldfield, P. Widener, A. B. Maccabe, L. Ward, and T. Kordenbrock, "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, Sep. 2006.
- [18] M. Folk, G. Heber, Q. Koziol, E. Pourmal, and D. Robinson, "An overview of the hdf5 technology suite and its applications," in *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*. ACM, 2011, pp. 36–47.
- [19] S. Byna, M. Chaarawi, Q. Koziol, J. Mainzer, and F. Willmore, "Tuning hdf5 subfilng performance on parallel file systems," 2017.
- [20] J. Lofstead, M. Polte, G. Gibson, S. Klasky, K. Schwan, R. Oldfield, M. Wolf, and Q. Liu, "Six degrees of scientific data: reading patterns for extreme scale science IO," in *Proceedings of the 20th international symposium on High performance distributed computing*, ser. HPDC '11. ACM, 2011, pp. 49–60. [Online]. Available: <http://doi.acm.org/10.1145/1996130.1996139>
- [21] C. Johnson, K. Keeton, C. B. Morrey III, C. A. Soules, A. C. Veitch, S. Bacon, O. Batuner, M. Condotta, H. Coutinho, P. J. Doyle *et al.*, "From research to practice: experiences engineering a production metadata database for a scale out file system," in *FAST*, 2014, pp. 191–198.
- [22] H. Sim, Y. Kim, S. S. Vazhkudai, G. R. Vallée, S.-H. Lim, and A. R. Butt, "Tagit: an integrated indexing and search service for file systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2017, p. 5.
- [23] J. Lofstead, F. Zheng, S. Klasky, and K. Schwan, "Adaptable, metadata rich io methods for portable high performance io," in *In Proceedings of IPDPS'09, May 25-29, Rome, Italy*, 2009.
- [24] R. Rew, E. Hartnett, J. Caron *et al.*, "Netcdf-4: Software implementing an enhanced data model for the geosciences," in *22nd International Conference on Interactive Information Processing Systems for Meteorology, Oceanography, and Hydrology*, 2006.
- [25] A. R. Thakar, A. Szalay, G. Fekete, and J. Gray, "The catalog archive server database management system," *Computing in Science & Engineering*, vol. 10, no. 1, 2008.
- [26] S. Albrand, T. Doherty, J. Fulachier, and F. Lambert, "The atlas metadata interface," in *Journal of Physics: Conference Series*, vol. 119, no. 7. IOP Publishing, 2008, p. 072003.
- [27] S. L. Pallickara, S. Pallickara, and M. Zupanski, "Towards efficient data search and subsetting of large-scale atmospheric datasets," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 112–118, 2012.
- [28] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proceedings of the 7th symposium on Operating systems design and implementation*. USENIX Association, 2006, pp. 307–320.