# SoMeta: Scalable Object-centric Metadata Management for High Performance Computing

Houjun Tang, Suren Byna, Bin Dong, Jialin Liu, and Quincey Koziol
Lawrence Berkeley National Laboratory, CA 94720, USA

***Abstract*—Scientific data sets, which grow rapidly in volume, are often attached with plentiful metadata, such as their associated experiment or simulation information. Thus, it becomes difficult for them to be utilized and their value is lost over time. Ideally, metadata should be managed along with its corresponding data by a single storage system, and can be accessed and updated directly. However, existing storage systems in high-performance computing (HPC) environments, such as Lustre parallel file system, still use a static metadata structure composed of non-extensible and fixed amount of information. The burden of metadata management falls upon the end-users and require ad-hoc metadata management software to be developed.**

**With the advent of "object-centric" storage systems, there is an opportunity to solve this issue. In this paper, we present SoMeta, a scalable and decentralized metadata management approach for object-centric storage in HPC systems. It provides a flat namespace that is dynamically partitioned, a tagging approach to manage metadata that can be efficiently searched and updated, and a light-weight and fault tolerant management strategy. In our experiments, SoMeta achieves up to $3.7X$ speedup over Lustre in performing common metadata operations, and up to $16X$ faster than SciDB and MongoDB for advanced metadata operations, such as adding and searching tags. Additionally, in contrast to existing storage systems, SoMeta offers scalable user-space metadata management by allowing users with the capability to specify the number of metadata servers depending on their workload.**

## I. Introduction

A revolution is in the making to handle the massive amount of data in extreme-scale computing systems. High-performance computing (HPC), inching towards exascale systems, enables scientific applications from various areas of physics, chemistry, materials, climate, etc., to generate data in the size of hundreds of terabytes and is projected to produce petabytes in the near future. To preserve the data for further analysis or even sharing among the community, various metadata, such as the data source and simulation configurations, are required to be stored and linked with the data. Existing HPC parallel file systems, such as Lustre [1], GPFS [2], and OrangeFS [3], face serious challenges in managing metadata as they only maintain the system metadata of data files, which is static and not extensible. Though self-describing data formats such as HDF5 [4], PnetCDF [5], and ADIOS [6] provide the functionality to keep metadata within the same data file, they focus more on data access optimizations and lack scalable metadata services, such as search and update. As a result, users often have to develop customized software for efficient and scalable metadata management.

Towards scalable data management on upcoming extreme-scale computing systems, researchers have proposed *objects* as the central feature of storage systems [7], [8], [9], such as OpenStack Swift [10] for cloud computing, and DAOS [11] for HPC systems. We refer to these storage systems as ***Object-Centric Storage (OCS)*** systems. As Arnold stated in [12]: "What makes an object storage from 'interesting' to 'transformative' is the ability to use the user-specified custom metadata as the basis for search queries". Having a metadata service in the OCS systems that provides efficient and convenient user-definable operations would greatly benefit scientific data users. However, various challenges must be addressed to achieve such a goal.

Managing the metadata of OCS systems effectively in a scalable manner is a critical requirement. Compared to the traditional block-based storage systems, we envision that future OCS systems will provide an abstract data model and semantically rich interfaces. In those systems, a data object contains data, such as multi-dimensional arrays and key-value pairs, while the metadata, stored as a metadata object, contains storage locations, data sources, or even the initial analysis results. Such metadata-rich usage scenarios will not sustain on existing file and directory-based hierarchical metadata management methods. The irrelevance, restrictiveness, and performance limits of methods from existing file systems have been well documented in literature [13], [14]. In preparation for OCS systems on upcoming exascale systems, it is vital for the metadata management to achieve scalability, extensibility, searchability, and fault tolerance.

Scalability for a HPC object-centric storage system is required to maintain the metadata associated with millions or even billions of data objects [15]. As users may add extra descriptive information dynamically, supporting extensible metadata is also a requirement. Locating data objects based on metadata requires efficient search mechanisms. A fault tolerant design is required to avoid any loss of data. A few systems have explored some of these aspects. For example, Ceph [8] proposed a scalable and fault-tolerant metadata management, while DeltaFS [16] experimented with a server-less metadata management approach. However, they are only applicable to existing hierarchical file systems, where files are organized by a directory tree. Database management systems, such as SciDB [17] and MongoDB [18] can be used for metadata management, but require manual data import, separate interfaces for maintenance, and are generally not optimized for the HPC environment. Recent research efforts, such as DAOS [11], are heading in the direction of managing data objects in an object-centric manner, however, DAOS metadata management component is still in the design phase and not yet available.

To address the above mentioned requirements, we propose SoMeta, a Scalable object-centric Metadata management method for HPC systems. SoMeta adopts a decentralized and

fault-tolerant approach to achieve scalable metadata operations. Metadata is separated from data objects as metadata objects, providing the flexibility that is ready to be integrated into OCS systems. The metadata objects are also extensible and searchable with our tagging approach, and is targeted for HPC with parallel computing workloads that have data structures and access mechanisms very different from typical database workloads. We will describe the design and an implementation of SoMeta, and evaluate the performance with a series of experiments. In specific, this paper has the following contributions:

- **Scalable flat-namespace metadata management for OCS systems.** SoMeta uses a flat namespace with inherent parallel support to distribute metadata objects, and allows an unlimited number of servers that are able to manage hundreds of millions of metadata objects efficiently.
- **A tagging approach for extensible and user-definable metadata.** Tags are key-value pairs that form a metadata object. SoMeta is designed to support dynamic tag creation, update, and delete operations.
- **Flexible metadata search support to retrieve interested metadata objects.** Users can search and retrieve metadata objects by using semantic tags, without having to remember object IDs that have limited semantic information.
- **A window-based adaptive fault tolerance mechanism.** SoMeta manages the metadata objects in memory to provide high performance. To handle any server failures, SoMeta periodically checkpoints the metadata to persistent storage devices, such as SSDs and hard disks. It is also able to recover from server failure at run time without data loss.

The remainder of this paper is organized as follows: In Section II, we present the design and the main components of SoMeta. We demonstrate the performance of SoMeta in Section III with basic metadata operations as well as advanced operations such as tagging and searching. We discuss the related work in Section IV and conclude the paper in Section V.

## II. Scalable Object-centric Metadata Management

### A. SoMeta Overview

SoMeta aims at providing a scalable and object-centric metadata management for HPC systems, and it treats metadata as individual *metadata objects*. Each metadata object contains basic identification information such as object ID, identification attributes, system information (e.g., creation time), as well as other user-defined attributes. This information is represented in the form of tags, which allows a user to form logical groups easily (detailed description is in Section II-D). These extensible and user-definable tags can be searched and modified efficiently through highly parallel SoMeta system, without the need of a centralized synchronization. In contrast to the hierarchical namespace used by traditional file systems, SoMeta adopts a flat namespace. Users can locate the interested data objects by searching for the tags. SoMeta can be easily integrated into an object-centric storage system as the metadata management component via our provided application programming interface (API). Additionally, we design our approach to provide high scalability on upcoming and future exascale systems.

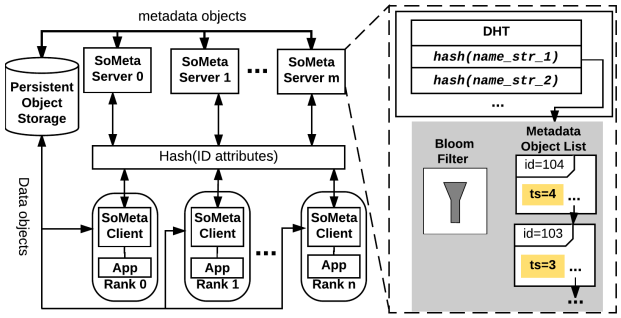


Fig. 1. Overview of SoMeta's architecture with $m$ servers and $n$ clients.

The two main components of SoMeta system, as shown in Fig. 1, are *SoMeta Servers* and *SoMeta Clients*. The servers store and manage metadata objects, and run in user space to enable scalability of metadata management. Applications, linked with the SoMeta Client library, communicate with the servers through network and perform operations such as creating, searching, updating, and deleting metadata objects. SoMeta partitions its flat namespace onto multiple servers using a Distributed Hash Table (DHT). Each entry of the DHT stores a list of the metadata objects that has the same name. Additionally, we use Bloom filters for accelerating the process of checking for metadata objects with the same name.

### B. Flat Namespace

A *flat namespace* is used by SoMeta to manage metadata objects, as opposed to the hierarchical directory approach used by existing file systems. We illustrate the two different approaches in Fig. 2, where a plasma physics simulation code (VPIC) produces multiple datasets with different time steps and configurations. In general, using a flat namespace avoids the overhead of traversing the prolonged directory path in the hierarchical namespace. However, a challenge of using flat namespace is due to the lack of logical organization (such as grouping by directory) for all metadata. Another challenge with flat namespace management is locating a specific metadata object or finding related metadata objects, which may be time-consuming because of the potentially large search space. We describe our solutions to these two challenges in Section II-C and II-E, respectively.

### C. Distributed Metadata Partition

Scalability is a critical requirement for metadata management in OCS systems to maintain good performance with a massive number of metadata objects. Managing all metadata objects on a single or a few metadata servers, similar to existing parallel file systems, will easily run into scalability problems.

In SoMeta, we distribute the metadata objects across multiple servers using a Distributed Hash Table (DHT), as shown in Fig. 1. DHT eliminates the need for a centralized server, and allows participating servers to efficiently locate a value (metadata object) associated with a given key (ID attributes).

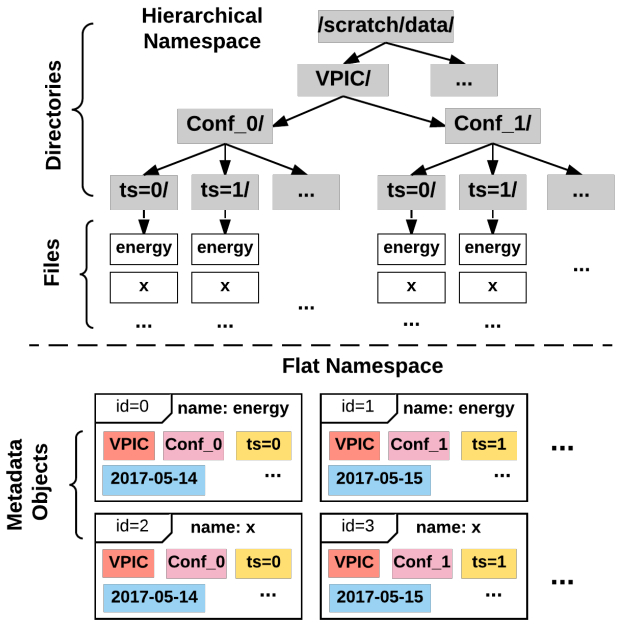


Fig. 2. An example to compare hierarchical and flat namespace. In the hierarchical namespace, files are organized into directories that form a tree. Locating the target file requires a traversal of the directory tree from the root. On the other hand, flat namespace has all metadata objects on the same level. Each metadata object can be located directly and independently. Tags, presented as colored boxes, are used in SoMeta for users to label data, which can then form logical groups and used for metadata search. Data objects are not included in this figure, while there is a one-to-one relationship between data and metadata objects.

Based on the estimated workload, i.e., the number of metadata objects, users can initiate a certain number of metadata servers in the background before their application starts. SoMeta uses a two level hash approach for efficient metadata access, with the first level partitioning the namespace among servers, and the second level of hashing will be used as hash key for hash table insertion. For example, in searching for a metadata object, SoMeta uses the following equation to find a Server ID for managing the object:

$$Server\ ID = HashFunction(ID\ attributes)\ \%\ p, \quad (1)$$

where $p$ is the number of SoMeta Servers, and ID attributes (defined in Section II-D) are concatenated into one string as the hash input. We chose to use multiple ID attributes instead of only the name of the object to avoid potential load imbalance when a large number of objects have the same name, but with varied time-step values, which is a common scenario for time-series simulation generated data. For the second level, only the name attribute is used as the key of the metadata. One reason of adopting this is to support efficient metadata search with incomplete ID attributes. For example, if the user wants to find all metadata objects of a specific name and between a time-step range, our name-only hash approach only needs to lookup one hash entry and iterate over the list of objects and check their time step values, as opposed to multiple lookup operations. While there are several hash functions available, in our implementation of SoMeta, we use the `djb2` hash function as it can result in balanced metadata distribution. We verify the load balance of the hash algorithm in Section III-G.

When adding a new metadata object to the DHT, a duplication check must be performed to identify all the objects with the same name. A full scan of the metadata objects is very costly, especially when a large number of objects have the same name. To alleviate this issue, we use a counting Bloom filter [19] to find an identical metadata object is definitely not in the set with constant time. Counting Bloom filter also supports removing items. The Bloom filter is created only when the number of objects in the metadata object list reach a configurable threshold. By default, we set this number to be 100. Additionally, we have found that the counting Bloom filter is able to achieve less than $0.1\%$ false positive rate, indicating $99.9\%$ of the time the full scan duplication check is avoided.

### D. Metadata Objects with Extensible and User-definable Tags

Each metadata object in SoMeta is a collection of tags, where the implementation of tags is based on a key-value model. Each tag is managed as a non-hierarchical key-value pair, as shown in Fig. 3. Users can add as many tags to an object depending on their desired content. When a tag is created by a user, it is mandatory to specify a string-typed key, while the value may be empty or may contain either strings or numeric values. Specifically, each metadata object has two types of tags: *predefined tags*, and *user-defined tags*.

- Predefined tags include *identifier (ID)*, the location of the associated data object, system information, and ID attributes. ID is a 64-bit integer to uniquely identify an object and can also be used to access an object directly. To guarantee the uniqueness, the IDs are generated by the metadata servers, and each server has a pre-defined range of valid ID values. The *system info* tag is similar to that of the Linux file system's *inode* data structure, containing *access control, creation, modification, last accessed, time stamps*, etc. Four *ID attributes* are selected to uniquely identify a metadata object. Specifically, the *object name* and *application name* are strings specified by user. The ownership tag contains the user ID, and *time step* is an attribute used to support time-series datasets. Hence, metadata objects can have same name as long as their other ID attributes are different.
- User-defined tags are an extensible list of key-value pairs. SoMeta provides the flexibility to allow any number of such tags created for each metadata object. By assigning tags, users can label a number of objects and form logical groups. As illustrated in Fig. 2, users can assign an "energy" metadata object with a tag of "Conf_0". Such an approach also avoids metadata duplication if a metadata object belongs to multiple groups.

The pre-defined tags are pre-determined and must be set either by the SoMeta or the user when creating an object, while user-defined tags are optional. For storage as well as network transfer, the metadata object is serialized by concatenating the tags with special split characters to an array.

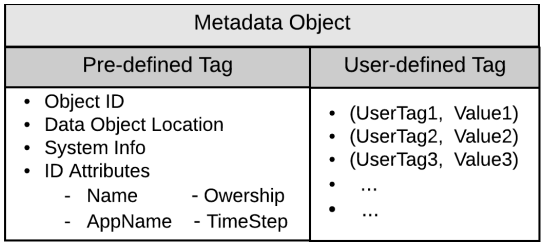


Fig. 3. Metadata object structure.

### E. Flexible Metadata Retrieval with Search

In SoMeta, there are two search methods to find and retrieve metadata objects: exact match search (**E-search**) and partial match search, (**P-search**). E-search requires all *ID attributes* provided by user, and is similar to the Linux `stat` command. P-search takes a conditional string on a subset of ID attributes and/or other tags, e.g, *"Name='VPIC' AND Conf=1"*, where *Name* is an ID attribute and *Conf* is a user-defined tag. All metadata objects that satisfy the constraint will be returned to the user, which can then be accessed with an iterator provided by SoMeta. P-search provides the feature similar to `find` or `grep` command, but it works in parallel and supports different search types such as string matching and value range query.

Note that SoMeta uses the ID-attributes to determine the target server and then uses the name string as the key of a hash entry in DHT. The Bloom filter is also helpful when a searched object does not exist. Otherwise, *E-search* needs to iterate over the metadata object list in finding a match. On the other hand, *P-search* allows the user to specify any tag of desired metadata objects. With incomplete ID attributes, the exact servers containing the searched objects cannot be determined. As a result, this type of search requires a client to send a search constraint to all the metadata servers. To speedup this process, SoMeta clients first coordinate among themselves and designate a number of clients that equals the number of servers, so that the search request is sent to the servers in parallel and each server communicates with only one client. When the object name is not included as part of the search constraint, all objects must be checked and it becomes the worst case scenario for SoMeta search. The search process can be improved by adding indexes, which is in development.

### F. Update and Delete

SoMeta supports adding, updating and removing tags. These operations require the object ID or all ID attributes of the target object. Once the desired metadata object is located on the target server, add or update operation is performed directly. However, due to the characteristic of DHT, an updated object may have to be moved to a different server if its ID attributes are changed. This process is done by inserting the new object to the corresponding server and then deleting the object on the original server. The *Delete* operation has a similar process as the update, once the object is found, it will be removed from the metadata object list, and the count of the Bloom filter is decreased, if the filter is present for the objects.

### G. Complexity Analysis

We provide the complexity analysis for each of the above-mentioned metadata operations in Table I. We denote $n$ as the total number of existing metadata objects, $p$ as the number of SoMeta Servers, $\delta$ as the name string's duplication ratio and $\delta = \frac{max\{c_i\}}{n}$, where $c_i$ is the count of $i^{th}$ name string. Therefore, the maximum length for the Metadata Object List for each entry of DHT table is $\delta n/p$. We also use $\alpha$ to denote the rate for Bloom filter to report uncertain existence. Theoretically, $\alpha = (1 - e^{-nk/m})^k$, where $k$ is the number of hash functions used by Bloom filter and $m$ is the number of bits in the array. Usually, $\alpha$ tends to be a very small value and the Bloom filter on SoMeta Server informs the non-existence of an object with a very high probability.

For creating metadata objects, SoMeta identifies the server ID using Equation 1. The duplicate check costs $O(1)$ if the Bloom filter finds that a duplicate does not exist. Otherwise, each item in the Metadata Object List will be checked. Therefore, the creation has the $O(\alpha\delta n/p)$ complexity on average and $T(\delta n/p)$ complexity in the worst case. As $\alpha$ is usually small, the average complexity is close to consistent time. The Update, Delete, and *E-search* operations has the same time complexity as the create operation because they have a similar process of communicating with a server and the follow up activity on the server.

*P-search* with object name has two cases: with or without a hit, as the Bloom filter identifies whether a hit is either possible in metadata object list or definitely not. When the search results in a hit, the time complexity for the average case is $O(\delta\frac{n}{p})$ as it needs to scan the list once anyway. On the other hand, the time complexity of *P-search* without a hit depends on the probability of positives (denoted with $\alpha$) of Bloom filter. In this case, the average time complexity of the name-tag based E-Search without a hit is $O(\alpha\delta\frac{n}{p})$, with $\alpha$ tends to be a very small value. The $O(\alpha\delta\frac{n}{p})$ can be approximately considered as constant. For *P-search* that does not have the name string as part of the input conditional string, it needs to search all metadata objects in parallel, therefore the complexity is $T(n/p)$.

TABLE I
COMPLEXITY ANALYSIS FOR SOMETA METADATA OPERATIONS.

| | Complexity | |
|---|---|---|
| | Average-case | Worst-case |
| Create | $O(\alpha\delta n/p)$ | $T(\delta n/p)$ |
| Update | $O(\alpha\delta n/p)$ | $T(\delta n/p)$ |
| Delete | $O(\alpha\delta n/p)$ | $T(\delta n/p)$ |
| E-search | $O(\alpha\delta n/p)$ | $T(\delta n/p)$ |
| P-search w/ name (no hit) | $O(\alpha\delta n/p)$ | $T(\delta n/p)$ |
| P-search w/ name (has hit) | $O(\delta n/p)$ | $T(\delta n/p)$ |
| P-search w/o name | $O(n/p)$ | $T(n/p)$ |

### H. Fault Tolerance

SoMeta manages the metadata objects in servers' memory, and a potential issue with this approach is that the metadata might be lost when one or more servers fail for any reason. To prevent such data loss, we designed SoMeta to use a window-based checkpoint-recovery mechanism. Specifically, the length

of the window is determined by a number ($s$) of write metadata operations. After a server's performing $s$ number of write operations, it will create a checkpoint to write all the metadata objects in its memory to the persistent storage. The persistent storage is adaptively selected between an SSD-based storage a disk-based storage, given their availability and capacity in the system. Meanwhile, it is possible that some metadata operations are performed between a server's checkpoint and failure time. Since it is very costly to checkpoint for every single operation, eacg SoMeta client maintains a linked list that records its last $s$ write operations, so that when a server failure occurs during the window interval, all the recorded operations are sent to the new proxy server for recovering.

To support runtime server failure/departure, each SoMeta Server is configured to have proxy candidates when it fails. Server failures are discovered by clients, when a SoMeta Client finds that a server does not respond within a specified time, it will broadcast to inform all clients to wait and mark that server as failure. The client then communicates with the failed server's proxy, with the default to be the failed server's $ID - 1$, waiting for it to read the checkpoint data of the failed server and reconstructs the DHT, then retrieves and performs metadata requests after the failed server's checkpoint time from clients. As the proxy server is now responsible for increased workload, we plan to explore and integrate dynamic load balancing strategy [20] into the SoMeta system.

### I. SoMeta Implementation and Usage

We have implemented SoMeta using C programming language. Users can configure SoMeta to manage specific storage nodes or to start the SoMeta on computing nodes of a HPC system. We focus on the latter case such that a user can start a number of SoMeta Servers suitable for their workload estimation. There are two modes that a user can run SoMeta Servers: **shared mode**, where the server process run on the same compute nodes as an application's MPI processes, and **dedicated mode**, where all server processes are on dedicated nodes that are separate from the nodes that run the application. For the former case, it is expected to perform better with more node-local communication between SoMeta server and clients. The user can also choose to have the servers running consistently, so the overhead of loading the metadata at start time is eliminated.

SoMeta Client is implemented as a linkable library and provides a set of metadata operation APIs. The client-server communication layer is implemented SoMeta's using Mercury [21], a C library for Remote Procedure Call and is optimized for HPC systems. In our experiments, we configured Mercury with BMI (communication component of OrangeFS [3]) plugin and using TCP protocol.

SoMeta does not maintain any replica at run-time, which avoids concurrent modification of replicas and thus achieves high performance. Each client's metadata request is sent to only one server, and it becomes straightforward to maintain the consistency.

TABLE II
EXPERIMENT CONFIGURATION.

| | Experiment Configuration |
|---|---|
| HPC Systems | Cori (Cray XC40), Edison (Cray XC30) |
| Comparison | Lustre, SciDB, MongoDB |
| Workloads | Synthetic (benchmark), Real-world application (BOSS) |
| Operations | Standard (create, delete, etc.), Advanced (add tag, search) |
| Storage | Hard disk drive, SSD-based Burst Buffer |

## III. RESULTS

### A. Evaluation Overview

We evaluated the performance of SoMeta with a series of experimental configurations, as shown in Table II. We compare the performance of SoMeta with Lustre file system, SciDB, and MongoDB, and used a standard benchmark-based workload, a synthetic workload, and an astronomy observational data analysis workload in our evaluation.

We run SoMeta systems on two supercomputers at the National Energy Research Scientific Computing Center (NERSC), named Cori and Edison. Cori is a Cray XC40 supercomputer with 1630 Intel Xeon "Haswell" nodes, where each node consists 32 cores and 128GB memory. On Cori, we tested SoMeta with both of its disk-based storage space and the "Burst Buffer", an SSD-based non-volatile storage space. Edison is a Cray XC30 system with 5500 Intel Xeon "Ivy Bridge" nodes, each with 24 cores and 64GB memory. Unless otherwise specified, we ran one SoMeta Server on each compute node (i.e., shared mode) in our tests to share resources with user applications.

Lustre is the most widely deployed HPC parallel file system. The tested Lustre file systems on Cori and Edison have a fixed set of nodes as metadata servers (MDS), which was set at system installation time. When comparing with Lustre, we used $mdtest$[1], a commonly used benchmark for traditional metadata operations, to compare the performance of standard metadata operations. The Lustre tests are performed when no other users are accessing the 4 MDSs. As SoMeta has many advanced abilities, such as adding user-defined tags and searching, are not directly supported by Lustre, we have created a synthetic benchmark with hand optimized code. We made our best effort to simulate object-centric metadata operations, such as file grouping and searching. For instance, we compare the search functions (i.e., E-search and P-Search) of SoMeta with hand-optimized code, which uses `stat` and `find` commands. We also use the "symlinks" as the metadata grouping counterpart implementation of Lustre to SoMeta tagging.

As SoMeta allows metadata objects with same names but different ID attributes, we used three types of synthetic workloads:

- **SoMeta 1**: all metadata objects have the same name but have different values in other ID attributes (e.g., time step).
- **SoMeta 4**: four unique object names are used and each name is used by a quarter of metadata objects. The objects with an identical name have different ID attributes.

[1] https://sourceforge.net/projects/mdtest/

- **SoMeta Unique**: each metadata object has a unique name.

We have also demonstrated the real world application's performance by applying SoMeta and using other technologies mentioned above to manage the metadata of Baryon Oscillation Spectroscopic Survey (BOSS). With the BOSS dataset, we also compared the metadata search function of SoMeta with SciDB and MongoDB, as they have been used manage array datasets and relational data sets, respectively.

### B. Metadata Creation

Creating the metadata is typically the first step in managing data in objects. In this experiment of metadata creation, we measured the number of object creations per second (throughput) in creating one million metadata objects. We used different numbers of client processes ranging from 120 to 3840 that issue creation requests concurrently. The number of servers increases with the number of clients at a fixed ratio and the total number ranges from 4 to 128 in both systems.

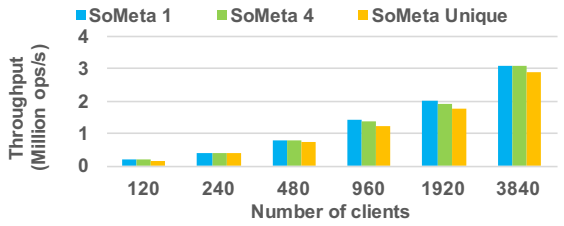


Fig. 4. Performance comparison for creating one million metadata objects on Cori.

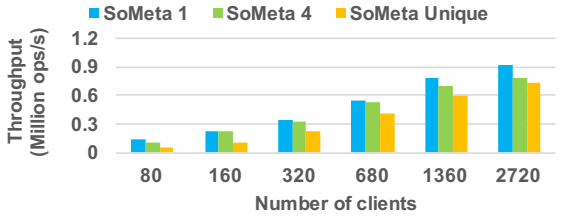


Fig. 5. Performance comparison for creating one million metadata objects on Edison.

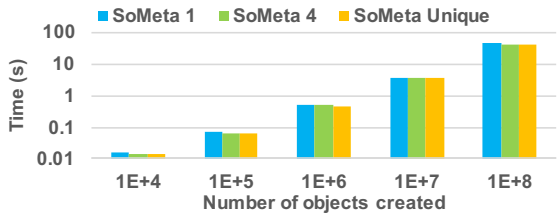


Fig. 6. Performance of scaling SoMeta system by creating up to 100 million metadata objects with 512 servers and 2560 clients on Edison.

Fig. 4 and 5 show the metadata creation throughput of SoMeta on Cori and Edison, respectively. Overall, they demonstrate the object creation throughput increases with the number of servers. Among the three workloads, SoMeta 1 yields the highest throughput, while SoMeta Unique has lower throughput. For SoMeta 1, all metadata objects have the same name but varied time step value, with our hash-based partition approach, they are distributed to all servers with a good load balance. Within each server, they stored in a linked list under one hash table entry, with an average length of $1000000/p$, and $p$ is the number of servers. Here the metadata object list is long enough to trigger the creation of the bloom filter, and the duplication check is accelerated. For SoMeta Unique workload, those objects are likely to be inserted into different hash entries (unless there is a hash collision). Hence, the metadata object list has few objects and the bloom filter is rarely created. We found that the hash entry insertion takes more time than inserting a metadata object into linked list plus maintaining and checking the bloom filter, and thus it is reasonable for SoMeta 1 to have the highest throughput and the SoMeta Unique being the slower one.

*Scalability tests on metadata creation.* Fig. 6 shows the performance of SoMeta with a fixed number of clients and servers, and the number of created objects increases from 10 thousand to 100 million. This is a stress test for SoMeta, and it is uncommon for a single user to create 100 million objects in one application run within a short time. However, from the figure, we can observe that with 512 servers, SoMeta is able to finish the creation of 100 million objects within 100 seconds. The time differences among the three workloads are also similar to the previous experiment. Due to space constraint, in the rest of the paper, we will report the results on Cori, as the results on Edison are similar.

### C. Metadata Search

As mentioned in Section II-E, SoMeta supports two types of metadata search: exact match search (*E-search*) and partial match search (*P-search*). In this section, we explored their performance characteristics with a set of experiments. Specifically, we tested these two search functions by varying the selectivity, i.e., the ratio of the number of the returned metadata objects satisfying a given conditional expression to the total number of metadata objects. Selectivity is expressed as a percentage and lower percentage number means fewer returned results for a search.

We used 128 servers in this test to maintain 1 million metadata objects. Among these objects, we varied the search selectivity by assigning tags to different metadata objects before the search, with the range of 2.5% to 20%. The time for completing *E-search* and *P-search* are reported in Fig. 7 and 8, respectively. We observe that both *E-search* and *P-search* take just a fraction of a second, even searching a large number of objects with a selectivity of 20%. Though *P-search* requires a full scan of all the 1 million objects, it takes less overall time to retrieve the same number of objects than *E-search*. This is because each *E-search* transfers only one metadata at a time, while *P-search* returns all the matched metadata to the client with one bulk transfer. The network latency of a large number of small data transfers dominates the overall time. Additionally, among three different tested workloads (from "SoMeta 1" to "SoMeta Unique"), SoMeta delivers almost equal performance. Therefore, it is reasonable to conclude that SoMeta provides a new and also efficient methods to perform metadata search, a 'transformative' feature in OCS systems.

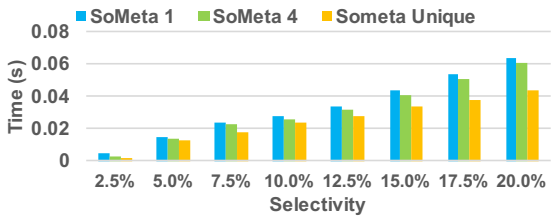


Fig. 7. Completion time of SoMeta's *E-search*.

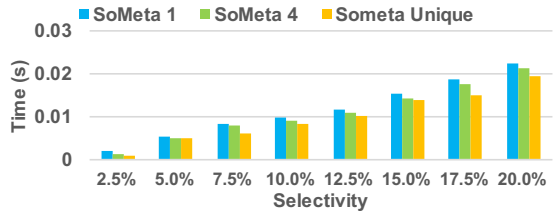


Fig. 8. Completion time of SoMeta's *P-search*.

### D. Metadata Update and Delete

In this section, we show the performance of updating and deleting metadata objects. The update targets are chosen randomly and the selectivity ranges from $2.5\%$ to $20\%$ of all the $1$ million metadata objects. $128$ server processes are used with the same number of clients sending the requests. The total time spent on updating different numbers of objects are shown in Fig. 9. We can observe that SoMeta has a very low latency in performing update operations. For example, in the case of $20\%$ selectivity, it takes less than $0.2$ seconds to finish updating all $1000000 \times 20\% = 200000$ metadata objects. For different workloads, "SoMeta 1" takes more time than the other two and "SoMeta Unique" offers the fastest response time. This is because of the search process that requires scanning the metadata object list in the DHT, where all objects of "SoMeta 1" are in the same linked list and need more time to find the match. The list of "SoMeta Unique" has much less objects and thus takes less time.

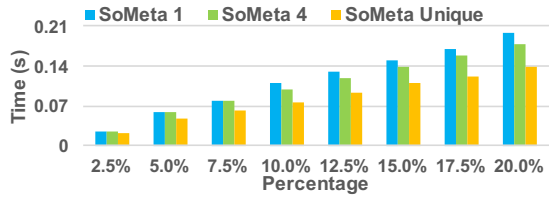


Fig. 9. Total time of metadata update.

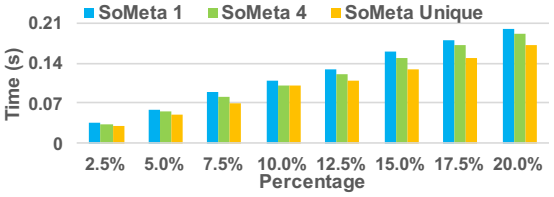


Fig. 10. Total time of metadata delete.

We measured the latency of deleting metadata object using the same configuration as that of the above update tests and the results are shown in Fig. 10. Similar to the update operation, deleting metadata objects requires finding the target object first, then removing it from the DHT and as well as the Bloom filter if it is created. It has a similar performance trend with metadata update. Among different workloads, the "SoMeta Unique" is the fastest one. The time to delete $200,000$ metadata objects only takes $0.2$ seconds.

### E. Comparison with Lustre

In this section, we compared SoMeta with the Lustre file system using *mdtest*, a popular metadata operation benchmarks for HPC storage system. We have configured the *mdtest* benchmark to perform traditional metadata operations, including file creation and deletion. While Lustre does not have an explicit search operation, in order to compare with SoMeta's search operation, we used `stat` and `find` functions from `glibc` and developed hand-optimized programs for locating target files in Lustre file system. The "stat" and "find" commands provide functions analogous to exact match and partial match searches of unique filenames using regular expressions. For Lustre, $1$ million empty files are created in different directories on $4$ MDS and $120$ clients. To have a fair comparison, although SoMeta can scale to much more number of servers, we have used the same number of metadata servers (i.e., $4$) for both systems in these tests.

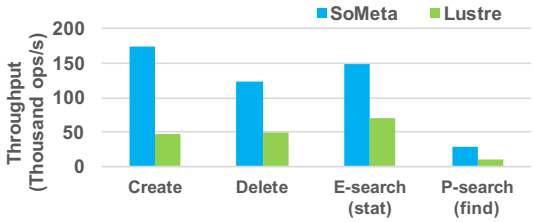


Fig. 11. A comparison of SoMeta and Lustre, where both systems use $4$ metadata servers, and accessed by $120$ clients.

As shown in Fig. 11, for metadata create and delete operations, SoMeta outperforms Lustre by $3.7X$ and $2.4X$, respectively. One major impact factor is that Lustre uses the directory tree-based namespace to organize its metadata, such that creating and deleting files need to access its parent directory all the way up to the root. In contrast, SoMeta uses flat namespace to organize the metadata objects that can be created and deleted directly. Since E-search and 'stat' go through the same steps as those of create, it is expected that SoMeta outperforms Lustre with 'stat'. For the 'find' operation, Lustre and other file systems require visiting all the files recursively under the provided directory and sub-directories. SoMeta's P-search is more efficient than Lustre+'find' since it does not require visiting all the objects unless an object name is not provided. Fig. 11 shows that SoMeta's E-search and P-Search outperforms Lustre+'stat' and Lustre+'find' by $2.1X$ and $2.6X$, respectively.

When we remove the fairness constraint of equaling the number of metadata servers with Lustre, SoMeta distributes

the metadata across all user-level server processes on different compute nodes. With unrestrained SoMeta using 128 metadata servers, we found out that SoMeta delivers $37X$ better throughput speedup over the current Lustre with 4 MDS. This high performance indicates that a scalable metadata management component as the one we had developed in this paper is a meaningful exploration for the future exascale storage systems.

### F. BOSS application

In this section, we report the results by using SoMeta to manage the metadata of a real-world application named Baryon Oscillation Spectroscopic Survey, or BOSS for short. The BOSS data is from the Sloan Digital Sky Survey (SDSS) project [2], and provides information about composition of stars and galaxies, and can be used to obtain their redshift, i.e., how fast a star is moving away from the earth. BOSS typically produces a single data file per object observed. Each BOSS data file is associated with three attributes, which can be represented as a triple *(plate, mjd, fiber)*, where *plate* is the SDSS plug plate ID that is used to collect the spectrum, *mjd* is the modified Julian date of the night when the observation was carried out, and *fiber* is the fiber number, ranging from 1 to 1000.

The dataset used in our tests includes $276,575$ files scattered in $4,888$ directories. One typical operation by the astronomers is to locate and access a subset of these files by providing a list of *(plate, mjd, fiber)* attribute triples. Since most existing parallel file systems do not support adding tags and searching on metadata, i.e., attribute triple for each file in this case, current BOSS management team develops their own querying program, which accepts the triple list and copies its matched files to a new concatenated file. Duplicating data is expensive but it can speedup future analysis of the same data and is also convenient for data sharing. We can also replace the expensive copy operations with symbolic links, which requires copying the system metadata and therefore can be more efficient. An alternative way to perform the metadata management for BOSS is to use database systems, e.g., SciDB and MongoDB. Potentially, loading the metadata of all $276,575$ BOSS files into these DB systems allow users to utilize the index technologies for faster query response. In the following paragraphs, we discuss and compare these methods with our SoMeta system.

**Metadata grouping comparison.** The process of selecting a number of interested data objects by a query constraint can be abstracted as forming logical groups of objects. SoMeta provides a natural solution for such a task through metadata search and tag update, so that the objects of a group can be easily accessed using a previously assigned tag. Existing parallel file systems such as Lustre do not support adding tags (e.g., user-defined attributes) to data files, the closest comparison for forming a logical group among files is to create symbolic links (symlink) for all interested files into a directory. We developed a hand optimized code that creates symlinks for the queried BOSS files in parallel. However, to use database management systems such as SciDB and MongoDB, the metadata must be first be converted to a specific format and then manually loaded into those systems. This takes extra time, ranging from seconds to minutes, and requires a separate interface for any following metadata operations.

[2]http://classic.sdss.org/legacy/index.html

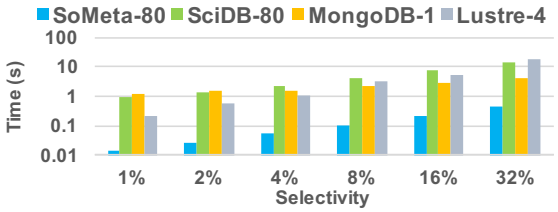


Fig. 12. Total elapsed time to group objects by adding tags (SoMeta), attributes (SciDB), symlink (Lustre) with different selectivity. The number after the name in the legend is the number of servers used.

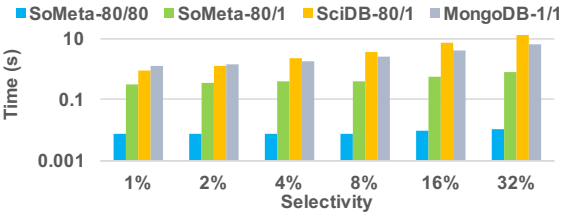


Fig. 13. Total elapsed time for searching and retrieving the metadata of previously assigned tags/attributes with different selectivity. The number after the method name indicates the number of servers and the number of clients.

Fig. 12 compares the total time for using SoMeta and other methods to group different numbers of objects with selectivity ranging from 1% to 32%, i.e., from $26,000$ to $832,000$ files or objects, respectively. For Lustre, as the queried files are scattered in different files and directories, the cost for using the hand-optimized code needs directory traversal as well as system metadata duplication, which leads to a relatively high overhead. SoMeta with 80 servers achieves $15X$ to $40X$ speedup over Lustre, which is expected as SoMeta stores individual objects within the flat namespace and therefore allows direct access without directory traversal. On the other hand, with SoMeta's parallel optimization for such workload, it is 10 to 90 times faster than SciDB and MongoDB.

**Metadata search comparison**. Fig. 13 compares the performance of using SoMeta, SciDB and MongoDB searching metadata. As mentioned previously, searching metadata of files requires users developing their own programs or using database systems, such as SciDB and MongoDB. Additionally, databases allow only one client to send a search request to their server, and thus are not optimized for typical HPC applications that distribute the workload to multiple processes and run in parallel. In comparison, our SoMeta is able to handle one search request issued collectively by multiple clients and therefore each process gets a share of the result.

To compare the performance, we have developed a program for importing all BOSS metadata to databases, where each record corresponds to one metadata object with a number of attributes. Both SoMeta and SciDB uses 80 servers and MongoDB uses only 1 server. All these methods are tested with *one* client sending the search request. In addition, we

TABLE III
STATISTICS ON THE DISTRIBUTION OF 1 MILLION METADATA OBJECTS ON 128 SERVERS.

| Statistics | SoMeta 1 | SoMeta 4 | SoMeta Unique |
|---|---|---|---|
| Minimum | 7812 | 7692 | 7575 |
| 1st Quartile | 7812 | 7782.25 | 7740 |
| Median | 7812.5 | 7812.5 | 7817 |
| 3rd Quartile | 7812 | 7842.75 | 7882.5 |
| Maximum | 7813 | 7933 | 8087 |
| Standard deviation | 0.50 | 85.19 | 99.19 |

also include the performance of SoMeta using 80 clients to issue the search request concurrently for most optimized performance. From Fig. 13, we observe that SoMeta delivers the best performance. SciDB does not allow creating an index for faster querying, and the query is executed by scanning all records regardless of selectivity. On the other hand, MongoDB allows users to create an index at data import time that can later be used to speed up metadata search. However, due to the configuration limit, only one MongoDB server was available to us, resulting it to be the slowest. SoMeta is $2X$ to $16X$ faster than SciDB and MongoDB when using *one* client, and up to $1000X$ with 80 clients. Theoretically, when running with one SoMeta server, the performance is expected to be slower than MongoDB. We expect a similar or better performance in single server tests after the integration of indexing technology to SoMeta in our future work. Overall, we can conclude that SoMeta provides a scalable and also efficient metadata search function for managing metadata objects on HPC systems.

### G. Load balance of SoMeta

Load balance among servers is another key factor for SoMeta to achieve high throughput by utilizing all available resource. Table III presents the statistics of the number of objects per server (i.e., workload) among 128 SoMeta metadata servers after creating 1 million objects. Overall, these metadata objects are distributed close to evenly as their quartiles and medians are very close to the mean value of $\frac{1,000,000}{128} = 7812.5$. With different workloads ("SoMeta 1" to "SoMeta Unique"), their distributions vary. In the "SoMeta Unique" case, even with 1 million objects of unique names, standard deviation still is relatively low, which indicates that SoMeta servers are able to achieve good load balance in common use cases.

### H. SoMeta Overheads

As a new metadata management system designed for HPC object storage system, we expect that SoMeta has ignorable overhead in system setup and less memory and storage footprint. Towards these goals, we measure and discuss the different overheads of our proposed system.

**Server initialization overhead.** In Fig. 14, we show the SoMeta servers' start time, which includes loading 1 million metadata objects from previously preserved metadata, reconstruct the DHT, and other preparations before accepting clients connections. The total amount of data related the metadata objects is $\approx 500MB$. As expected, the majority of the startup

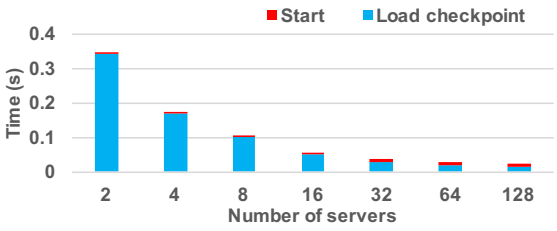


Fig. 14. Overhead in loading one million metadata objects from checkpoint file into memory.

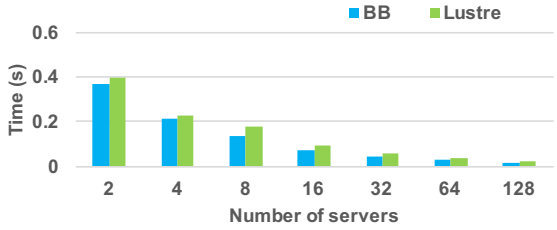


Fig. 15. Total time spent in checkpointing 1 million objects onto Burst Buffer (BB) and Lustre file system.

time is spent on reading the data from a storage system, yet the overhead remain under half a second in all cases. As the amount of metadata is relatively small, the overhead does not decrease significantly with more than 16 servers. The total time increases with more servers due to the global synchronization for all servers at start time that guarantees all are functioning correctly.

**Persistent storage overhead.** To support fault tolerance, SoMeta checkpoints the metadata from the servers' main memory to the persistent storage system. Fig. 15 shows the total time spent on checkpointing 1 million objects with various numbers of SoMeta servers. It takes less than 0.4 seconds in all cases to write the checkpoint data to the persistent storage. Burst buffer (BB) takes less time than Lustre, but not by a significant amount as we expected. We believe it is because only small-sized I/O is performed, since less than $500MB$ of data is written by all servers.

**Server memory usage.** SoMeta stores metadata in memory to provide fast response for various metadata operations. We show that the memory consumption overhead by all participating server in Table IV, after creating different numbers of metadata objects, ranging from $50k$ to 1 million. These low amounts of memory overhead is a minute fractions of total available memory on the compute nodes ($64GB$ on Edison and $128GB$ on Cori) we ran our experiments, and has minimal impact on user applications.

## IV. RELATED WORK

In the database domain, SciDB [17] uses a flat namespace based on PostgresSQL to manages its metadata (e.g., array schema). We have tested the performance of metadata operations, but due to its single metadata instance and therefore poor scalability, a fair comparison is not feasible, and thus was not included in our paper. Existing NoSQL databases, such as MongoDB [18] and Cassandra [22] are generally not optimized for HPC environment because most HPC workloads

TABLE IV
SOMETA SERVER MEMORY USAGE.

| Objects | SoMeta 1 | SoMeta 4 | SoMeta Unique |
|---|---|---|---|
| 50000 | 35 MB | 36 MB | 37 MB |
| 100000 | 63 MB | 69 MB | 72 MB |
| 500000 | 337 MB | 345 MB | 356 MB |
| 1000000 | 643 MB | 685 MB | 713 MB |

are write intensive from tens of thousands of CPU cores. To the best of our knowledge, there is no scalable way to directly ingest HPC data from simulation codes into a DBMS, as demonstrated by [23].

The tree structure-based hierarchical metadata management methods are widely adopted in single-node or parallel/distributed file systems. General examples for the single-node file system include NTFS, EXT4, Btrfs [24]. Some specific single file systems (such as JFFS2 [25]) are developed for flash SSD devices. Distributed file systems such as HDFS [26] and GFS [27] use a single node to manage the whole hierarchical namespace. In order to improve the performance of metadata operations, most recent distributed file systems such as Ceph [8], Lustre [1], LH [28], OrangeFS [3], and FusionFS [29] partition the whole directory tree among multiple servers. However, since the hierarchical namespace requires traversing directory tree path from root before accessing the actual data, extra overhead is involved to traverse the directory tree and leads to potential multi-pass communications to different servers [30]. In addition, it is reported Ceph [8] does not scale well in HPC environment [31]. MetaKV [32] is a specialized key-value store that indexes the file segments on distributed burst buffers. Each file segment is a one-dimensional region of a shared file. BurstMem [33] and TRIO [34] use stacked AVL tree to index file segments on burst buffers. The generic property graph [35] is proposed to manage and organize the data, but such graph-based methods may share the similar issue as the tree-based methods in supporting efficient partitioning among multiple nodes.

Different from the hierarchical namespace, the flat namespace is proposed to manage all metadata on a single level. Typical examples of these flat namespace systems are Copernicus [14] and hFAD [13]. Some scientific data formats (like HDF5 [4] and ADIOS [36]) allow users to store metadata with their data in the same file. However, the issues of scalable metadata operations such as tagging and searching have not yet been fully addressed. Developing ad-hoc metadata management system methods for different applications is another main efforts towards efficient metadata storage systems. Blanas et al. [37] provide a full survey of these methods. For example, Atmospheric Data Discovery System (ADDS) [38] was developed to store, index and search the observed atmospheric dataset and metadata. While adopting the flat namespace, our proposed SoMeta supports tagging and searching to efficiently label and group objects, as well as retrieve those of interest with a highly parallelized architecture. In addition, SoMeta allows users to directly operate on metadata objects, without the need to touch data objects and ensures scalability.

The Object-based storages (e.g., NASD [39], T10 [7], PanFS [40]) usually treat the data and their attributes together as a whole data object. As the attributes are stored and transferred with the objects, object-centric storage can efficiently express quality-of-service, transparent performance optimizations, data sharing, and data security qualities that storage system can exploit. Different from those approaches, our work treats the metadata itself as individual objects, that records the location, group, and other system information of the corresponding data object.

## V. CONCLUSION AND FUTURE WORK

Existing HPC file systems are inefficient and not scalable when managing user metadata. They are not capable of supporting operations such as searching the metadata. Aiming at future object-centric storage systems on HPC systems, where extensive metadata management is supported, we have developed a scalable metadata management infrastructure, named SoMeta. SoMeta views the metadata as separate objects to provide a flexible way for users and storage systems to maintain the extensive information about data. Different from the hierarchical tree-structure based namespace used by existing HPC systems, SoMeta uses a flat namespace to organize metadata objects. The flat namespace within the SoMeta is partitioned and served by multiple user-level server processes for extremely scalable and concurrent operations. We introduced a tagging method to label objects and form logical groups of related objects. To locate and retrieve interested metadata objects efficiently, a parallel search approach is introduced with the utilization of data structures such as Bloom filters to accelerate the search process. We have demonstrated the scalability of SoMeta, which is $3.7X$ faster than Lustre file systems in standard metadata operations and up to $16X$ faster than SciDB and MongoDB for searching metadata objects. In the near future, we plan to integrate with SoMeta upcoming object-centric storage systems and optimize performance further such as using indexing technologies to accelerate the search performance.

## ACKNOWLEDGMENT

This work is supported by the Director, Office of Science, Office of Advanced Scientific Computing Research, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231 (Project: Proactive Data Containers, Program manager: Dr. Lucy Nowell). This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

## REFERENCES

[1] P. J. Braam *et al.*, "The lustre storage architecture," 2004.
[2] F. B. Schmuck and R. L. Haskin, "Gpfs: A shared-disk file system for large computing clusters." in *FAST*, vol. 2, 2002, pp. 231–244.
[3] M. Moore, D. Bonnie, B. Ligon, M. Marshall, W. Ligon, N. Mills, E. Quarles, S. Sampson, S. Yang, and B. Wilson, "Orangefs: Advancing pvfs," *FAST poster session*, 2011.
[4] The HDF Group. (1997-) Hierarchical Data Format, version 5. Http://www.hdfgroup.org/HDF5.

[5] 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 *Proceedings of the 2003 ACM/IEEE Conference on Supercomputing*, ser. SC '03. New York, NY, USA: ACM, 2003, pp. 39–.

[6] J. F. Lofstead, S. Klasky, K. Schwan, N. Podhorszki, and C. Jin, "Flexible io and integration for scientific codes through the adaptable io system (adios)," in *Proceedings of the 6th international workshop on Challenges of large applications in distributed environments*, ser. CLADE '08. New York, NY, USA: ACM, 2008, pp. 15–24.

[7] M. Factor, K. Meth, D. Naor, O. Rodeh, and J. Satran, "Object storage: the future building block for storage systems," in *Local to Global Data Interoperability - Challenges and Technologies, 2005*, June 2005, pp. 119–123.

[8] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, ser. OSDI '06. Berkeley, CA, USA: USENIX Association, 2006, pp. 307–320.

[9] A. Devulapalli, D. Dalessandro, P. Wyckoff, N. Ali, and P. Sadayappan, "Integrating parallel file systems with object-based storage devices," in *SC '07*. New York, NY, USA: ACM, 2007, pp. 27:1–27:10.

[10] J. Arnold, *Openstack swift: Using, administering, and developing for swift object storage*. " O'Reilly Media, Inc.", 2014.

[11] J. Lofstead, I. Jimenez, C. Maltzahn, Q. Koziol, J. Bent, and E. Barton, "Daos and friends: a proposal for an exascale storage system," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Press, 2016, p. 50.

[12] J. Arnold, *Object Storage for Genomics Deploying and administering OpenStack Swift with SwiftStack for bioinformatics*. SwiftStack, 2015.

[13] M. Seltzer and N. Murphy, "Hierarchical file systems are dead," in *Proceedings of the 12th Conference on Hot Topics in Operating Systems*, ser. HotOS'09. Berkeley, CA, USA: USENIX Association, 2009, pp. 1–1.

[14] A. Parker-Wood, D. D. E. Long, E. L. Miller, M. Seltzer, and D. Tunkelang, "Making sense of file systems through provenance and rich metadata," University of California, Santa Cruz, Tech. Rep. UCSC-SSRC-12-01, Mar. 2012.

[15] S. Patil and G. Gibson, "Scale and concurrency of giga+: File system directories with millions of files," in *Proceedings of the 9th USENIX Conference on File and Stroage Technologies*, ser. FAST'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 13–13.

[16] Q. Zheng, K. Ren, G. Gibson, B. W. Settlemyer, and G. Grider, "Deltafs: Exascale file systems scale better without dedicated servers," in *10th Parallel Data Storage Workshop*, ser. PDSW '15, New York, NY, USA, 2015, pp. 1–6.

[17] P. G. Brown, "Overview of SciDB: Large Scale Array Storage, Processing and Analysis," in *ACM SIGMOD*, 2010, pp. 963–968.

[18] MongoDB. MongoDB. Https://www.mongodb.com/.

[19] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese, "An improved construction for counting bloom filters," in *European Symposium on Algorithms*. Springer, 2006, pp. 684–695.

[20] B. Dong, X. Li, Q. Wu, L. Xiao, and L. Ruan, "A dynamic and adaptive load balancing strategy for parallel file system with large-scale i/o servers," *J. Parallel Distrib. Comput.*, vol. 72, no. 10, pp. 1254–1268, Oct. 2012.

[21] J. Soumagne, D. Kimpe, J. A. Zounmevo, M. Chaarawi, Q. Koziol, A. Afsahi, and R. B. Ross, "Mercury: Enabling remote procedure call for high-performance computing," in *2013 IEEE International Conference on Cluster Computing, CLUSTER 2013, Indianapolis, IN, USA, September 23-27, 2013*, 2013, pp. 1–8.

[22] 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.

[23] S. Blanas, K. Wu, S. Byna, B. Dong, and A. Shoshani, "Parallel data analysis directly on scientific file formats," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 2014, pp. 385–396.

[24] O. Rodeh, "B-trees, shadowing, and clones," *Trans. Storage*, vol. 3, no. 4, pp. 2:1–2:27, Feb. 2008.

[25] C. h. Chen, W. t. Huang, C. t. Chen, and R. s. Hsiao, "Improving the reliability of jffs2," in *2006 International Symposium on VLSI Design, Automation and Test*, April 2006, pp. 1–4.

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

[27] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, ser. SOSP '03. New York, NY, USA: ACM, 2003, pp. 29–43.

[28] S. A. Brandt, E. L. Miller, D. D. Long, and L. Xue, "Efficient metadata management in large distributed storage systems," in *Mass Storage Systems and Technologies, 2003.(MSST 2003). Proceedings. 20th IEEE/11th NASA Goddard Conference on*. IEEE, 2003, pp. 290–298.

[29] D. Zhao, Z. Zhang, X. Zhou, T. Li, K. Wang, D. Kimpe, P. Carns, R. Ross, and I. Raicu, "Fusionfs: Toward supporting data-intensive scientific applications on extreme-scale high-performance computing systems," in *Big Data (Big Data), 2014 IEEE International Conference on*. IEEE, 2014, pp. 61–70.

[30] X. Li, B. Dong, L. Xiao, L. Ruan, and D. Liu, "Cefls: A cost-effective file lookup service in a distributed metadata file system," in *CCGrid '12*, May 2012, pp. 25–32.

[31] F. Wang, M. Nelson, S. Oral, S. Atchley, S. Weil, B. W. Settlemyer, B. Caldwell, and J. Hill, "Performance and scalability evaluation of the ceph parallel file system," in *Proceedings of the 8th Parallel Data Storage Workshop*. ACM, 2013, pp. 14–19.

[32] T. Wang, A. Moody, Y. Zhu, K. Mohror, K. Sato, T. Islam, and W. Yu, "Metakv: A key-value store for metadata management of distributed burst buffers," in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2017, pp. 1174–1183.

[33] T. Wang, S. Oral, Y. Wang, B. Settlemyer, S. Atchley, and W. Yu, "Burstmem: A high-performance burst buffer system for scientific applications," in *Big Data (Big Data), 2014 IEEE International Conference on*. IEEE, 2014, pp. 71–79.

[34] T. Wang, S. Oral, M. Pritchard, B. Wang, and W. Yu, "Trio: burst buffer based i/o orchestration," in *Cluster Computing (CLUSTER), 2015 IEEE International Conference on*. IEEE, 2015, pp. 194–203.

[35] D. Dai, R. B. Ross, P. Carns, D. Kimpe, and Y. Chen, "Using property graphs for rich metadata management in hpc systems," in *Proceedings of the 9th Parallel Data Storage Workshop*, ser. PDSW '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 7–12.

[36] Q. Liu, J. Logan, Y. Tian *et al.*, "Hello ADIOS: the challenges and lessons of developing leadership class I/O frameworks," *Concurrency and Computation: Practice and Experience*, vol. 26, no. 7, pp. 1453–1473, 2014.

[37] S. Blanas and S. Byna, "Towards exascale scientific metadata management," *CoRR*, vol. abs/1503.08482, 2015. [Online]. Available: http://arxiv.org/abs/1503.08482

[38] S. L. Pallickara, S. Pallickara, and M. Zupanski, "Towards efficient data search and subsetting of large-scale atmospheric datasets," *Future Gener. Comput. Syst.*, vol. 28, no. 1, pp. 112–118, Jan. 2012.

[39] G. A. Gibson, D. F. Nagle, K. Amiri, J. Butler, F. W. Chang, H. Gobioff, C. Hardin, E. Riedel, D. Rochberg, and J. Zelenka, "A cost-effective, high-bandwidth storage architecture," *SIGPLAN Not.*, vol. 33, no. 11, pp. 92–103, Oct. 1998.

[40] B. Welch, M. Unangst, Z. Abbasi, G. Gibson, B. Mueller, J. Small, J. Zelenka, and B. Zhou, "Scalable performance of the panasas parallel file system," in *6th USENIX Conference on File and Storage Technologies*, ser. FAST'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 2:1–2:17.