SAND2020-1451C

SystemX: Turbocharging Scientific Investigation Through Comprehensive Metadata Management

Anonymous Author(s)

ABSTRACT

One of the biggest impediments to discovery for large-scale scientific applications is that they produce very large datasets, which are costly to load and search in their entirety. In this paper, we present SystemX[1], which provides the following solution: users can insert extensible, user-defined metadata attributes so that they can rapidly determine which data subsets are "interesting" and should be loaded for further analysis. SystemX offers generalized metadata encoding and querying techniques; independent, portable metadata; scalable metadata consistency techniques; fault-tolerance as a service; flexible system configuration; and high performance and scalability.

KEYWORDS

[blinded for review]

ACM Reference Format:

Anonymous Author(s). 2020. SystemX: Turbocharging Scientific Investigation Through Comprehensive Metadata Management. In *HPDC'20: The 29th International Symposium on High-Performance Parallel and Distributed Computing, June 23–26, 2020, Stockholm, Sweden.* ACM, New York, NY, USA, 12 pages. https://doi.org/XXXX

1 INTRODUCTION

One of the greatest obstacles to making discoveries from large-scale scientific applications and large scientific instruments is that they produce immense volumes of data that are difficult to store, manage, and explore efficiently. Simulations such as S3D combustion [8], XGC edge plasma fusion [16] and GTS core plasma fusion [28], and data collection instruments such as the LSST [13] and Square Kilometer Array Radio Telescope [10], and genome sequencing [12] can produce datasets in the terabytes to petabytes range in a single day. As we move towards exascale, these data volumes will continue to increase as scientists run simulations of increasing fidelity and deploy instruments with more sensitive sensors.

In the past, data volumes have been small enough that scientists could load entire datasets during post-processing to search for interesting data. However, with the large datasets being produced today, scientists can no longer afford the wasted node hours or the time delay associated with retrieving uninteresting data from storage only to discard it for further analysis. Instead, scientists need an efficient way to identify and load only the interesting data.

[1]Name anonymized for review

HPDC'20, June 23–26, 2020, Stockholm, Sweden

ACM ISBN XXXX…$15.00
https://doi.org/XXXX

Custom metadata offers a promising approach. Scientists can perform lightweight, in-situ analysis to identify interesting features such as a combustion event, storm cell, or area of high turbulence, tag these events, and then, during post-processing, use this metadata to load only the data associated with these features. Restricting reads to this "interesting" data can result in significant speedups, thereby accelerating analysis and discovery. An example of this workflow is depicted in Figure 1.

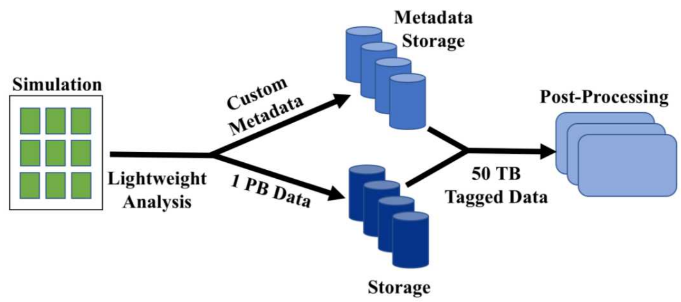

Figure 1

Despite this need for a custom metadata management, existing solutions are relatively primitive. While popular I/O libraries offer the ability to attach metadata attributes to variables or files within a dataset, they do not offer any technique to accelerate accessing these attributes, such as indexing, or a way to query only a subset of the attributes. With hundreds of thousands or millions of processes all generating metadata, performing a linear-time search across the stored metadata attributes results in a significant drag on productivity. Further, it is not possible to attach an attribute to a variable subset, limiting the use of these tools in HPC where a single variable can be up to a few petabytes in size. Analysis is accelerated by reducing the reading scope, and if a metadata tool cannot facilitate this, it will not significantly improve productivity. Finally, these approaches embed metadata within the variable or file, making it very costly to perform global searches since the entire data hierarchy must be linearly traversed and searched to retrieve the associated metadata. Other solutions have offered more robust querying capabilities, but at the cost of usability and performance. These solutions have required users to learn domain specific languages or specialized querying languages such as SQL and keep track of unique identifiers, or relied on non-indexed, flat namespaces that must resort to scanning all stored metadata to be able to provide the full range of queries that users need. Merely providing the ability to flexibly add and query metadata is not sufficient. The system must be reasonably easy to use and must ensure the provided metadata services are high-performing and scalable. Moreover, none of these proposed solutions offer atomic operations, limiting their ability to be used concurrently such as in a workflow.

In this paper we introduce SystemX, which offers the custom metadata solution that users need, and which addresses all of the

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & 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.

concerns listed above. Through previous work [citations blinded for review], we developed the following features, which have been incorporated into SystemX:

- **Generalized Metadata Encoding Techniques**. These techniques enable SystemX to encode features of different type, scope, and datatype using a consistent queryable format.
- **Generalized Metadata Querying Techniques**. These techniques give users an easy-to-use, scalable, optimized mechanism to filter attributes based on any of their characteristics, including their associated run, timestep or variable, feature type, value, and logical spatial location.
- **Independent, Portable Metadata**. SystemX allows metadata to be loaded and explored independently of the data without losing the connection between metadata and data or compromising metadata portability.

One of the contributions of this paper is presenting improved versions of the following features:

- **Scalable Metadata Consistency Techniques**. These techniques ensure metadata consistency for any workflow without compromising performance and scalability.
- **Fault-Tolerance as a Service**. SystemX provides fault-tolerance as a service, allowing users to decide what trade-off of fault-tolerance and performance best suits them.

This paper further extends our previous work by providing the following features:

- **Flexible System Configuration**. SystemX allows users to easily adjust the system based on their functionality requirements, available resources, and priorities.
- **High Performance and Scalability**. SystemX provides high performance and scalability in each of its provided services. In addition, SystemX allows users to further tune performance by providing tools to identify and mitigate performance and scalability bottlenecks.

The rest of the paper is organized as follows. In Section 2 we will build off of the ideas presented here to develop a concrete set of requirements for a custom metadata management system. In Section 3 we present our custom metadata management solution called SystemX. We discuss SystemX's design and how it meets the system requirements. Section 4 presents a brief overview of the implementation of SystemX that is evaluated. Section 5 contains the testing and evaluation information. Section 6 presents related work, and Section 7 discusses future work.

2 SYSTEM REQUIREMENTS

In the previous section, we established that HPC scientists need a custom metadata management solution to ease and accelerate analysis and exploration. In this section we explore more deeply what functionality a custom metadata management solution must provide to meet scientist's needs. These features include: flexible metadata attributes; robust metadata queries; usability; independent, portable metadata; metadata consistency; system reliability and availability; system flexibility; and high performance and scalability.

2.1 Flexible Metadata Attributes

A robust custom metadata management solution must allow users to store any kind of metadata, and in a way that is meaningful to them. One aspect of this requirement is that users must be able to associate attributes with different components of their datasets such as an entire run, timestep, or variable or a subset of a variable. Users should also be given complete flexibility in their attribute type (how they name the type of feature) and in the data type of the associated value. For example, a scientists running GTS will need to tag a spatial area in one or more variables to indicate the presence of a "blob" while a S3D scientists may wish to tag an entire application run as producing a combustion event. The system should offer a general solution by providing domain-independent metadata management. This will ensure users can store all of their metadata in one place.

2.2 Robust Metadata Queries

A custom metadata management solution must provide a wide range of metadata queries to ensure that users can easily and efficiently retrieve the subset of metadata attributes they are interested in. Beyond being able retrieve attributes associated with a particular run, timestep, or variable, users should be able to retrieve attributes of a particular type, associated with a particular value, and in a particular logical spatial area. For example, an LSST scientist may wish to query for all supernova events that have occurred in the past year or to hone in on a single image and retrieve a list of all known objects in that image. A GTS scientist may wish to retrieve all areas of turbulence that are located near the reactor edge. Finding a way to efficiently support this wide range of queries is one of the system's greatest challenges.

2.3 Usability

An important requirement that is often overlooked is usability. The system should be easy to use, minimize user burden, and allow users to store their metadata in a way that they find meaningful. Users should not have to learn a domain specific language or querying language to store and retrieve their metadata. In addition, they should not be required to remember particular identifiers to be able to access and make sense of their metadata. Finally, users should be shielded from the system's implementation. Users should not have to be familiar with the implementation to use the system correctly or to adjust their usage of the system if the implementation changes.

2.4 Independent, Portable Metadata

For a custom metadata management system to be able to accelerate scientific exploration, it must allow users to download their metadata, explore it locally to identify what data they want to analyze further, and then load only this "interesting" data. Thus, a critical requirement is that the metadata be decoupled yet tightly integrated with the raw data so users can quickly map from the metadata to the associated file or bytes of data. In addition, it is important for this metadata to be portable to other storage systems and layouts since scientific data is often moved between storage tiers or shared across storage systems. Users should not have to update each piece of metadata every time the data storage changes to be able to map from their metadata to the associated data. This would not only place a substantial burden on users but would also easily result in coherence issues.

2.5 Metadata Consistency

A custom metadata management solution must provide consistency in the face of concurrent accesses. The system might be shared by workflow components, applications, or even users, and must have a mechanism to ensure the integrity, accuracy, and completeness of the stored metadata despite this simultaneous usage. Users should be able to determine when their metadata is considered complete and correct and when it should be made externally visible. In addition, if users discover an error with their stored metadata, they should be able to correct it.

2.6 Reliability and Availability

A robust custom metadata management solution must offer both reliability and availability. With any system, faults are inevitable. However, users should be shielded from these faults whenever possible. The system should be able to quickly recover from a wide range of errors, guarantee the accuracy of results despite these errors, and provide uninterrupted service. Users should also be able to indicate a preference for more or less robust fault-tolerance since fault-tolerance often entails a performance penalty.

2.7 System Flexibility

It is important to recognize that no one system will be optimized for all use cases. Scientific applications can exhibit very different characteristics and can be run on very diverse hardware. In addition, users may have vastly different priorities when it comes to factors such as performance and resiliency. A system should be able to accommodate these priorities and to ensure that it gives users the functionality they need without penalizing them for functionality they do not need.

2.8 High Performance and Scalability

In HPC, where individual application runs can use millions of processes, performance and scalability are essential. Core hours are a precious resource, and any service that uses these resources and slows down application runs must try and minimize its impact. In addition, since a custom metadata management system's primary goal is to accelerate discovery, any performance limitations will detract from the system's ability to accomplish this goal. The system must thus offer services that perform and scale well for a single server, and provide the ability to scale out the service to meet demand.

3 DESIGN

In this section we present SystemX, our custom metadata management solution, which was designed to meet the system requirements we laid out in the previous section (see Section 2). This section will provide an overview of SystemX's features and will explain how these features tie in with the system's contributions. SystemX has been in development for over two years, and some of its features have been touched on by previous work [citations blinded for review]. These features include: developing generalized metadata encoding and querying techniques, and providing usability. We have also built off of this previous work to provide scalable metadata consistency techniques and fault-tolerance as a service. Finally, in this paper we introduce new features of flexible system configuration, and high performance and scalability.

3.1 Generalized Metadata Encoding Techniques

SystemX supports domain independent, extensible, user-defined metadata using a conceptual metadata model that can be seen in Figure 2. Users can insert basic and custom metadata for runs, timesteps, variables, and subsets of variables, which are high-level constructs that application scientists are accustomed to. Basic metadata captures the structure of a simulation output and simple information about the various components. Custom metadata refers to user-defined metadata attributes, which can be used to highlight interesting features in the associated data. Users can insert metadata of any datatype. Each custom metadata attribute is associated with a single tag (named label), which can be used to filter for particular kinds of metadata attributes. Using a metadata model that can simultaneously support domain independent and extensible, user-defined metadata is one of SystemX's more important contributions.

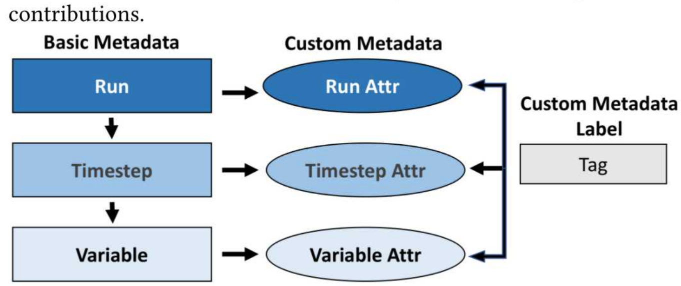

Figure 2

3.2 Generalized Metadata Querying Techniques

SystemX offers a rich, programmatic query interface that allows users to perform a wide range of optimized queries through a set of predefined API calls. These operations are robust, and users can perform even more complex queries by using a series of API calls. Users can filter attributes based on the run, timestep or variable they are associated with, the attribute's tag (type), the associated particular value, and the logical spatial location of the attribute. These queries are designed to accelerate analysis, and in particular, global, spatial, temporal, and multi-variate analysis. Being able to offer such a wide range of efficient, scalable metadata queries is another important SystemX contribution.

3.3 Usability

SystemX has a strong emphasis on usability. It shields users entirely from domain specific languages and querying languages such as SQL, and instead provides a programmatic API. In addition, SystemX does not require users to generate unique names for their metadata, allows users to query based on meaningful concepts such as the name of an application rather than system-generated IDs, and provides functions so that users do not have to remember the names or structures of their applications. This high level of usability is an important contribution of SystemX.

3.4 Independent, Portable Metadata

SystemX keeps the stored metadata decoupled from the associated raw data to ensure that users can download and explore their metadata without having to load the associated data. Once users have used local metadata queries to identify what data they want to explore further, SystemX allows them to map to the associated data in a portable way. Instead of storing a direct link to a data location in a file or object, SystemX annotates based on the location in a logical space, such as the global simulation domain. This idea is explored in [citation blinded for review]. Allowing users to efficiently explore their metadata and then map to the associated data in a portable way is one of SystemX's most important contributions.

3.5 Scalable Metadata Consistency

SystemX offers scalable, atomic operations to help ensure the integrity, completeness, and availability of stored metadata. These atomic operations are offered through three options for transaction management, which are designed for different use cases.

3.5.1 Method 1. The first is a variant of the open-source D^2T [9, 18] doubly distributed transactions system. The system allows users complete flexibility in what metadata objects are grouped into a transaction, the number of concurrent transactions, and if and when to commit or abort a transaction. This means the same set of metadata servers can safely be used for both the compute and post-processing components of a workflow. However, a limitation of this approach is that, since it stores transaction information for each piece of metadata, adjusting the visibility of a transaction requires scanning all stored metadata attributes in addition to updating the visibility of each adjusted attribute. Therefore, this transaction system will not scale well.

3.5.2 Method 2. The second transaction system maintains a separate metadata store for each ongoing transaction and for the set of committed transactions. This transaction method improves transaction scalability, since the cost of adjusting the visibility of a transaction is O(transaction size), and may improve write performance, depending on the implementation, since writes are to a relatively empty metadata store. Downsides include a temporary increase in storage overheads, a possible decrease in transaction performance since the metadata must be copied to the "committed" location, and an inability to change the visibility of a transaction once it has been committed. This system is thus best for when users want the flexibility of the D^2T system but with improved write performance and scalability, and can afford the increased transaction management cost.

3.5.3 Method 3. The third system limits the flexibility of the transactions in an attempt to significantly increase transaction performance. Limitations of the approach are that simultaneous transactions may not be possible (depending on the implementation), which limits the metadata server's ability to serve multiple applications simultaneously. An additional limitation is that, with many implementations, reads that must see only the committed transactions (e.g., reads in workflows), cannot occur while a write transaction is ongoing. Since write performance should be high, this should be a minimal burden. However, this limited availability might not work for all applications. Further details on these transaction management systems can be found in a publication in preparation. The second and third transaction methods, which were developed for this paper, offer solutions to the potential performance and scalability bottleneck of the D^2T system, and contribute to SystemX's goal of offering scalable atomic operations.

3.6 Fault-Tolerance as a Service

SystemX ensures system reliability and availability through fault-tolerance. SystemX provides flexible, scalable fault-tolerance, designed to allow the user to decide what level of resiliency they wish to use and how to respond to different failures.

3.6.1 Service Availability Discovery. SystemX deploys its metadata servers dynamically. This allows the service to grow and shrink with demand and provides resiliency in the face of server failures. Since SystemX's servers are dynamically deployed they require a discovery mechanism. For this reason, SystemX offers a directory service that processes can query to get a list of currently available servers. This is very similar to the placement groups employed by Ceph [29].

3.6.2 Durability. If, for performance reasons, the metadata is stored in memory, the system will be vulnerable to data-loss in the event of a hardware or software failure. To improve durability, SystemX provides a function that allows users to checkpoint the database to disk. Users can checkpoint the database more or less frequently depending on their needs. SystemX offers three different checkpointing modes, each of which is designed for a different scenario. The first mode involves keeping all metadata in memory and checkpointing to a single file. This is the only mode that allows servers to have access to all metadata for answering read queries (such as in a workflow scenario), and minimizes the need to do file compaction (combining multiple checkpoint files to produce a smaller number of files). The next mode keeps only non-checkpointed metadata in memory and checkpoints to a single file. This reduces the writing and checkpointing costs (since the database size is smaller) without increasing the compaction overhead. This mode is thus preferable if users do not need to perform queries across all stored metadata and want a single checkpoint file either to serve as a signaling mechanism or to minimize compaction costs because their parallel file system experiences significant bottlenecks. The last method keeps only non-checkpointed metadata in memory and produces a separate checkpoint file per database. This further reduces checkpointing costs but increases compaction costs and increases the pressure on the parallel file system's metadata server(s). This mode is ideal if hardware failures are unlikely (meaning a checkpoint file is unlikely to be used) or for parallel file systems with ample metadata servers. Further details on these checkpointing systems will be discussed in a publication in preparation. These checkpointing methods provide a solution to the potential scalability bottleneck created by checkpointing, and help SystemX maintain reliability.

3.6.3 Recovery. SystemX provides return values for all of its functions. This allows users to detect metadata storage related errors. SystemX also offers functions that allows users to delete or adjust the visibility of metadata. This allows users to decide how they wish to respond to any errors that occur.

3.7 Flexible System Configuration

As discussed in the requirements section, flexibility is critical to ensuring that users have access to the functionality they need, without having performance penalties for features they do not need. In addition to the transaction management and checkpointing options described above, SystemX offers four different run modes, offering users two choices for a service mode and two choices for a storage mode.

3.7.1 Service Modes. SystemX offers two service modes: the dedicated mode and the embedded (or local) mode. The architecture for the dedicated mode can be seen in Figure 3. The dedicated mode performs metadata management outside of the context of the client application(s) using a set of dedicated server processes. Users, either directly or through a user interface such as an I/O library, call functions from the SystemX API library. The SystemX client then sends a message to a SystemX server, which performs the requested metadata interaction and sends result back to the client. As discussed above in Section 3.6, the system uses a directory service to ensure the service is discoverable. The service is visible to multiple applications enabling simultaneous use and may or may not be part of the storage system. In contrast, the embedded mode, also known as the "local" mode, manages all metadata in the compute nodes using node-local memory for metadata storage. The architecture for the local mode can be seen in Figure 4. The SystemX client manages all metadata locally, thus eliminating the need for message passing.

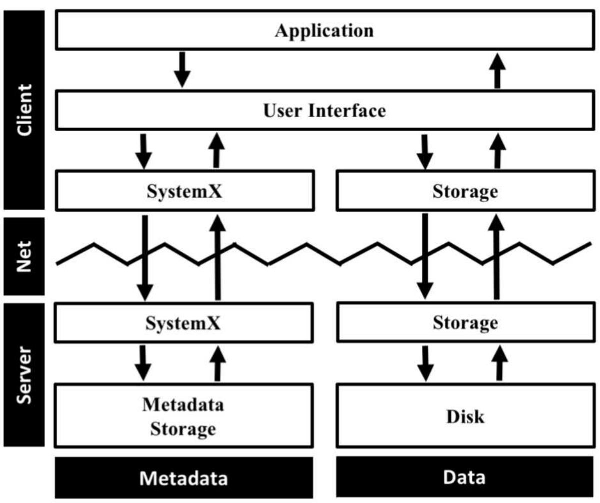

Figure 3

The dedicated mode offers a number of advantages. The dedicated mode can minimize the impact on the application. The embedded mode requires node-local memory and compute time that could otherwise be used by the application whereas the dedicated mode uses the memory of the dedicated server nodes and can use asynchronous operations to require virtually no compute time from the compute nodes. The dedicated mode also makes it easier to offer workflow support since the metadata is stored outside of the compute nodes, and accessing this metadata will thus not disturb the application. In addition, when the metadata is stored in memory, the dedicated mode offers improved durability since there are fewer

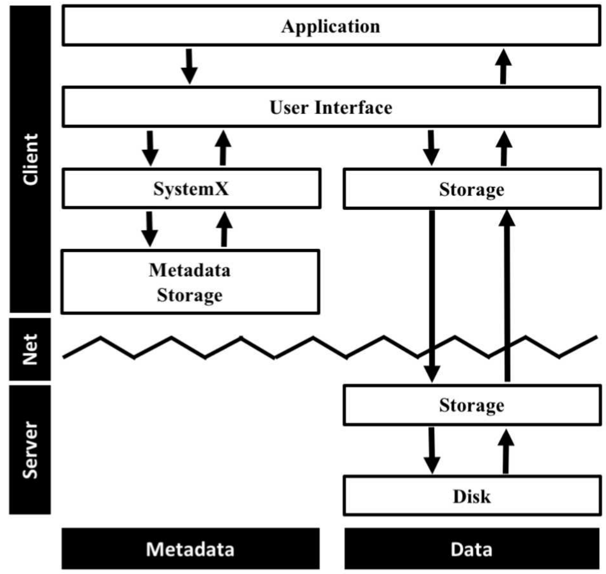

Figure 4

metadata stores and thus fewer points of failure. Finally, if the results of read queries need to be globally distributed, the dedicated mode will offer better performance since it will require broadcasting the query results provided by each server, which is O(number of servers), whereas the embedded mode will require an all-to-all for the compute processes, which is O(number of compute processes).

The embedded mode offers a number of advantages as well. The embedded mode should offer much higher performance for writes and for read queries that do not need to be globally shared since there is no need for message passing or possibility of a metadata server bottleneck. Checkpoints may be faster as well since each process will checkpoint a much smaller amount of metadata. However, if these files need to be compacted to produce a reduced number of total checkpoint files, this will reduce the performance significantly. The embedded mode should also experience better scalability with respect to the number of client processes. We would expect write-performance to stay constant, the amount of available RAM to scale linearly, and read and checkpoint performance to scale linearly or sub-linearly (depending on the implementation). This mode thus ensures good scalability without the dedicated resource requirement of the dedicated mode. Finally, the embedded mode is simpler and less fragile because it does not require a server discovery mechanism or server communication mechanism. Having a directory service introduces a single point of failure, and the reliance on the supercomputer's network introduces additional failure possibilities.

3.7.2 Storage Modes. SystemX offers two storage modes: in-memory and on disk. The primary advantage of storing the metadata in memory is better expected performance since memory tends to be orders of magnitude faster than disk. However, as we will see in the Evaluation Section (see Section 5), it is not always the case that both writing and reading are faster for the in-memory case due in part to the presence of caches. Storing on disk offers a number of advantages. First, there is much greater capacity when storing on disk. Nodes tend to have much more disk capacity than memory capacity due to the cost of RAM (and the relative newness of NVMe devices).

For the dedicated mode, the servers may be memory bound, and thus a reduced storage capacity will increase the number of server processes we must allocate, taking away resources that could otherwise be used by the application. Storing on disk also offers a significant durability advantage, since, in the event of a crash, the in-memory store will lose all metadata that has not been checkpointed to disk. Finally, not having to checkpoint to disk makes the on-disk case simpler, eliminates a significant performance cost, and provides better availability since, for the in-memory case, the servers will be unavailable as they perform the checkpoint.

3.7.3 Conclusions. Overall, there are many pros and cons of each run mode, as will be further highlighted in the Evaluation Section (see Section 5). These run modes are critical to allowing SystemX to offer scalable distributed metadata management, scaling out either with the number of compute processes or dynamically allocated dedicated servers. They also contribute to SystemX's flexibility, allowing users to decide whether to prioritize write performance (using the local, in-memory mode), read performance (using the dedicated, in-memory mode) or durability (using the on-disk mode). These modes also allow users to adjust SystemX depending on the availability of resources, such as using local mode if they have limited access to additional node allocations or on-disk mode if nodes have little available RAM.

3.8 High Performance and Scalability

. One of the most important components of a custom metadata management solution is that it be high-performing and scalable. While we have endeavoured to make all of the metadata services described above high-performing, we discovered a few potential scalability bottlenecks. Here we discuss our solutions to bottlenecks related to indexing and bottlenecks experienced in the dedicated service case, which can experience delays due to message passing and server bottlenecks.

3.8.1 Indexing. While indices are critical to performing complex reads efficiently, they can introduce a significant write penalty since each write must also update the index. Indices also dramatically increase storage overheads. In addition, many applications do not perform significant reads and thus do not need indices until post-processing if ever. SystemX provides users with the option to delay the creation of indices until writing has concluded or to never create them. These options contribute to SystemX's flexibility and help eliminate a large performance penalty and potential scalability bottleneck.

3.8.2 Synchronicity. One limitation of the dedicated service mode is that clients might spend a long time waiting for responses from the servers. Costs include message passing, both for sending a request and receiving a response, waiting for the server to respond to the request, which could result in a significant delay in the event of a server bottleneck, and time for the server to perform the metadata interaction. This could result in a significant waste of compute time, particularly for operations such as checkpointing the database to disk, which merely need to inform the clients that they were successful. For this reason, SystemX offers both synchronous and asynchronous versions of each function. The advantage of using the synchronous functions is that it is simpler for the user. The user does not have to keep track of which functions have completed and what values she is expecting in return from each function. The advantage of the asynchronous functions is that, for the dedicated service mode, they offer a significant performance increase since they allow users to overlap compute node operations with metadata server operations. These asynchronous functions are an important part of SystemX's flexibility and offer a solution to a performance bottleneck since, with hundreds or thousands of compute processes assigned to a single metadata server, there is often a significant server bottleneck.

3.8.3 Message Bundling. An additional potential bottleneck for the dedicated mode case is having a large number of small messages. If each client sends a separate write or read request, most messages both to and from the metadata server could be under 1 KB in size. This results in large messaging overheads and under-utilization of network bandwidth. This is particularly problematic since, if users minimize the resources allocated to the metadata servers, they will likely experience a significant metadata bottleneck. SystemX offers two solutions: message bundling for a single client and message bundling across multiple clients. SystemX provides functions that allow clients to bundle write requests, so that multiple pieces of metadata can be written in a single request. SystemX also provides functionality to allow users to perform global write or read requests, relying on well-tuned MPI collectives. Queries can be funneled to a subset of clients that combine the requests into a single message and then, if necessary, distribute the results. A final optimization is that SystemX uses RDMA for large message transfers, improving the effective bandwidth. Message bundling is yet another component of SystemX's flexibility and can help relieve a metadata server bottleneck by bundling together requests.

4 IMPLEMENTATION

SystemX uses an RDBMS backend for metadata storage. This allows the system to offer its wide range of scalable, efficient metadata queries. SystemX uses a set of distributed, shared-nothing servers, that each maintain a copy of the basic metadata and a horizontal shard of all user-defined attributes. This allows for distributed query processing with minimal overheads (by minimizing server-side coordination). When dedicated servers are used, by default, client processes are distributed evenly across the available servers. This can be adjusted by the user if they have additional information about load balancing or if they wish to implement a metadata distribution and querying mechanism that is optimized for their application.

4.1 Using SystemX

An early version of SystemX is available at [blinded for review]. Once the full system has finished export review, it will be available at the same address. SystemX's functionality will be exposed to the user as a C++ library.

4.2 External Libraries

Our evaluated implementation of SystemX uses SQLite 3.27.2 [1] as the metadata storage backend, the GNU C++ compiler version 8.2.1 and OpenMPI 1.10. SQLite is chosen because it is open-source, and because of its server-less model, dynamic type system, and

Cluster	Nodes/Cores	Proc. Type	OS	Intercon.	RAM per Core
ClusterA	1488/53,568	2.1 GHz Intel Broadwell	RHEL 7	Omni-Path	3.5 GB
ClusterB	1122/40,392	2.1 GHz Intel Broadwell	RHEL 7	Omni-Path	3.5 GB

Table 1: Compute Clusters used in Testing

light-weight design. Since the system will need to be installed on clusters, having a database with a serverless architecture is critical. Although containers offer a potential workaround, they add complexity both to the metadata management system and for the user's application. SystemX uses Faodel [27] for message passing between the clients and servers. Faodel is built upon the long stable and performant NNTI RDMA communication layer from the Nessie [21] RPC library from Sandia and provides asynchronous message passing and message queuing. SystemX 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.

5 EVALUATION

Our previous work [citation blinded for review] offered evidence that metadata management can significantly accelerate data analysis with trivial storage overheads by allowing users to rapidly identify data subsections that are of interest and load only these areas. Here we evaluate whether SystemX offers the performance and scalability needed for a production-oriented HPC metadata management system.

5.1 Testing Environment

Testing is performed on the ClusterA[2]capacity cluster at [institution name blinded for review] and utilizes the Lustre parallel file system. Tests are also run on the ClusterB[2]capacity clusters at [institution name blinded for review] but show similar results and are omitted for space considerations. Information about the two clusters can be found in Table 1.

5.2 Testing Configurations

To compare SystemX to alternatives, we evaluate the metadata management that can be implemented in HDF5. We use HDF5 version 1.10. HDF5 was chosen since it is the most frequently used I/O library for HPC science applications [6] and thus offers a realistic representation of the metadata management available to scientists today. In addition, HDF5 offers superior metadata management to other commonly used I/O systems since it offers scoped attributes and user-defined datatypes for attributes. More details on how we used HDF5 to provide most of SystemX's features can be found in [citation blinded for review].

We perform scalability tests of SystemX and the HDF5 comparison system using 1000, 2000, 4000, and 8000 processes for writing. Apart from this, all tests use 8000 write processes. The dedicated server tests use one tenth as many processes for reading (100, 200, 400, 800), 1 metadata server per 1000 write clients (1, 2, 4, 8), and use the same number of servers for the reading phase (1, 2, 4, 8). The embedded server tests use the same number of processes for reading (1000, 2000, 4000, and 8000) since all client processes have a shard of the metadata. Each testing configuration is performed a minimum of five times, and results are averaged across these runs. The one exception is the HDF5 comparison system runs, where each testing configuration is performed a minimum of three times rather than five (due to job timeout issues and resource constraints). Last-first timing is used (where possible), meaning that timing measures the time that passes between the first process that reaches task A and the last process that completes task B.

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). The testing harness is composed of three main parts: writing, reading, and checkpointing.

5.3 Writing

Each test simulates an application writing metadata for 1000 timesteps, where each timestep is composed of a set of 10 3D variables. Variables used in this evaluation include temperature, pressure, and density. Each of these variables is distributed across the processes using a 3D domain decomposition, so that each process writes metadata for a regular hyper-rectangle (a "chunk") for each variable. 10 different types of custom metadata attributes are written, each of which has a set frequency that determines what percentage of chunks it is associated with. These types 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. On average, 2.6 attributes are written per chunk (per variable, per timestep). In all, this amounts to over 200 million metadata attributes distributed across the 8 server processes for the 8000 write client case. 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.

5.3.1 *Reading and Checkpointing.*

Every 100th timestep, a set of read queries are performed to evaluate how performance varies as the metadata volume per server increases. Reading consists of 2 stages. The first stage performs six read patterns that are identified by the Six Degrees of Scientific Data[19] as typical for analysis codes. These six patterns are, for a given timestep:

(1) Read all data
(2) Read all data for a variable
(3) Read all data for 3 variables
(4) Read a plane in each dimension
(5) Read a 3D subspace
(6) Read a partial plane in each dimension

[2]Name anonymized for review

However, instead of reading the data for these patterns, the clients retrieve the associated metadata attributes for a particular tag. This reflects use cases such as using SystemX to rapidly identify data of interest, summarize global trends or provide high-level sampling statistics. These six read patterns are performed first for a tag that appears on 25% of all data, then 5% of all data and then .1%. This provides evidence of SystemX's ability to perform spatial queries efficiently. In the second stage of reading, the clients perform one global, one temporal and one multivariate query. After these reading stages conclude, each database is checkpointed (if the database is not on disk). Thus, with 1000 timesteps, reading and checkpointing are performed 10 times each.

5.4 Results

5.4.1 Scalability. Figure 5 demonstrates the scalability for writing, transaction management, and checkpointing the database to disk for SystemX's default configuration: using dedicated servers, the D^2T [18] transaction system, non-delayed database indexing, and a checkpointing method that copies the entire database disk each time. As we can see, the system achieves good scalability. The small differences in performance for writing and the larger difference in performance for the transaction management can be attributed to stragglers. Since the number of clients per server is held constant, this close to constant performance is what we would expect. We can also see there is a slight increase in the time needed for checkpointing. This is likely due to contention for the parallel file system resources (the metadata servers and disks). Since the databases will likely be checkpointed asynchronously, this performance difference is unlikely to matter. However, we could likely improve the checkpointing performance for each individual server by staggering slightly when they perform their checkpointing.

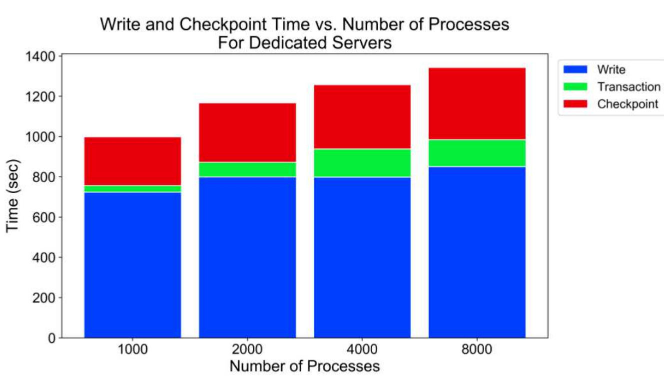

Figure 5

5.4.2 Comparison to Alternatives. As we can see in Figure 6, which uses logarithmic scaling on the y-axis, there is large performance difference between SystemX and the HDF5 comparison system. For our system, since we store our metadata in-memory, we include the additional cost of checkpointing the database to disk (although, this could be performed asynchronously by the servers). We can see that our system obtains better write performance (which will be discussed more below), and substantially better read performance. Our reads are performed in 2.61 seconds whereas is takes the HDF5 comparison system 51193.12 seconds. This difference is due in large part to our ability to scale out metadata operations (whereas HDF5 requires them to be serialized), and SystemX's ability to use extensive indexing of the metadata. This is a feature HDF5 is hoping to offer in the future. Overall, this read performance difference reflects how RDBMS's offer well-tuned data access methods out-of-the box, and how this is a feature I/O systems are lacking when it comes to metadata management.

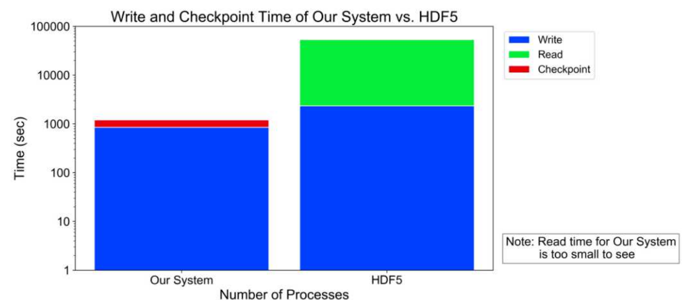

Figure 6

In Figure 7 we can see how SystemX's write performance scales compared to HDF5's. Whereas SystemX can scale out, increasing the number of metadata servers, HDF5 cannot. With fewer than 2000 write clients, HDF5 actually performs better since it performs a gather and single write whereas SystemX performs one small write per client. However, SystemX is able to maintain more-or-less constant write performance by scaling out to maintain a constant client-server ratio (1000:1) while HDF5 is not. As applications scale up to use hundreds of thousands or even millions of processes, using a single process to write application metadata will no longer be tenable.

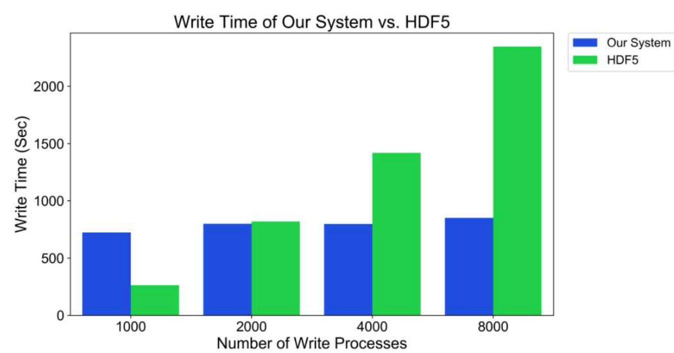

Figure 7

5.4.3 Service Modes. As mentioned above, SystemX offers two service modes: dedicated (using dedicated processes to serve solely as metadata servers) and embedded (managing the application metadata using the compute processes). The performance of these two models is compared in Figure 8. The precise timing results are summarized in Table 2. From the graph, we can see that using the local service mode substantially outperforms the dedicated service mode. Writing is significantly faster since there is no need to send small messages across the network and no chance of a metadata server bottleneck. Transaction management is significantly faster since each transaction and individual database is smaller. Checkpointing is faster since, again, each database being checkpointed is smaller. Finally, the local servers do not have to spend time compacting

	Write	Trans-action	Read	Checkpt	Compact
Dedicated	851.56	133.82	2.61	358.79	193.56
Local	2.63	2.09	7.69	32.10	0.00

Table 2: Runtime Comparison for Dedicated and Local Service Modes

the databases since the total metadata volume per process is small enough it can easily fit in memory. However, it is important to note that, as we can see in the table, reading takes almost 3X as long for the local servers. This is because it requires global coordination across all write processes to both read, and share the results of the read. This reflects one of the main limitations of the local server case: that it is more complex and slower to perform reads across the entire set of metadata since the metadata is significantly more distributed.

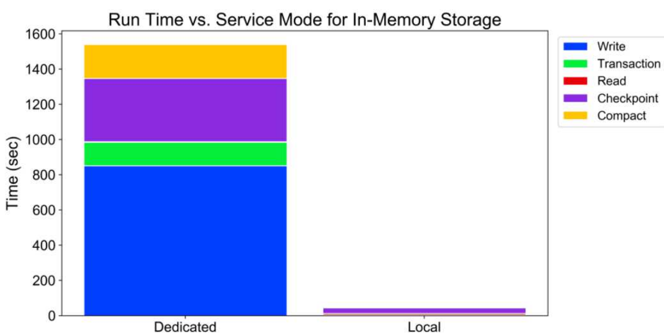

Figure 8

5.4.4 Storage Modes. Figures 9 and 10 compare how the different storage modes (storing the metadata in-memory vs. on disk) perform for the dedicated and local service modes, respectively. As we can see in both figures, the on disk case performs significantly worse than the in-memory case, due to both writing and transaction management being significantly slower. Since these test cases use SystemX's default transaction management system (the D^2T [18] system), which updates the transaction status of each piece of metadata, transaction management requires performing a full table scan and a significant update of the table. Using the in-memory storage mode incurs an additional cost of having to checkpoint the database to disk and, in some cases, compacting the checkpoint files to produce a single checkpoint file per database. For the local service mode, the amount of metadata per compute process will typically be small enough that all metadata can be kept in memory, thereby eliminating the need to produce multiple checkpoint files and thus the need to compact these files. We can see that in both cases checkpointing results in a moderate cost, and for the dedicated server case, compacting results in an additional cost. However, storing the metadata in-memory still offers a large performance advantage. In addition, it is important to remember that, for the dedicated service mode, checkpointing will likely be done asynchronously by the servers, and thus will likely overlap with compute phases on the client processes and will not affect the client's run time.

Since the read times are small enough that they are difficult to compare, they are displayed in Figure 11. From this figure we can see that, as expected, reading from disk is significantly slower than reading from memory for the dedicated case. However, for the local case, reading performance is approximately the same for on disk and in-memory. This is likely due to the fact that the total metadata volume per compute process is small enough that it can be maintained in SQLite's "page-cache", which is stored in main memory. We know that caching does not produce this result since we flush the cache before the start of each timestep.

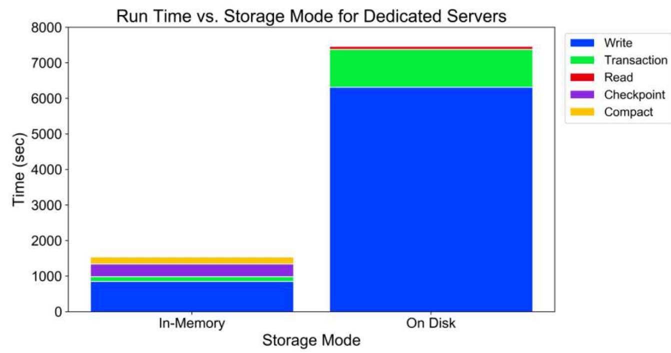

Figure 9

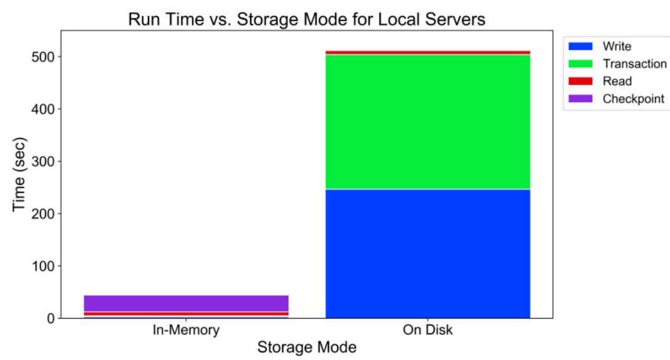

Figure 10

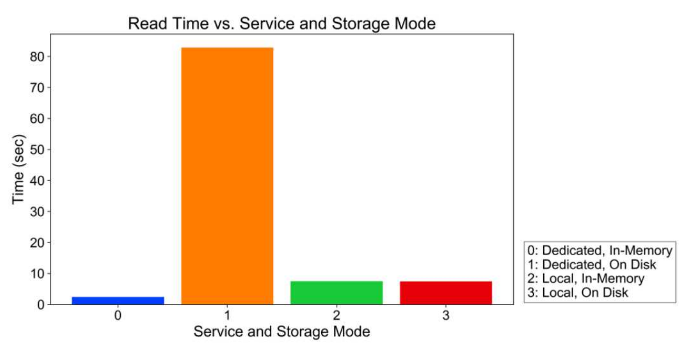

Figure 11

5.4.5 Checkpointing. Figures 12 and 13 compare the performance of the implemented checkpointing methods for the dedicated and local service mode cases, respectively. A summary of the various checkpointing schemes can be found in Table 3. Methods 0 and 1 involve keeping all metadata in-memory (when possible) and

	Single Checkpt. File	Multiple Checkpt. Files
Full DB	Method 0	
In memory	Method 1	
Partial DB	Method 4	Method 2
In Memory	Method 5	Method 3

Table 3: Checkpoint Methods

checkpointing to a single file. Methods 2-5 do not keep metadata in-memory after it has been checkpointed to disk. Methods 2 and 3 checkpoint to a separate file per checkpoint whereas methods 4 and 5 checkpoint to a single file. We can see that, for the dedicated servers case, methods 4 and 5 perform best. By keeping only un-checkpointed metadata in memory and checkpointing to a single file, these methods eliminate the need to perform database compaction. Methods 0 and 1 have to compact the database because they attempt to keep all metadata in-memory, and run out of RAM. Methods 2 and 3 utilize a separate file per checkpoint and thus will always require checkpoint file compaction. We can see that it is possible for methods 2 and 3 to perform similarly to methods 4 and 5. However, as we can see with the performance of method 2, methods 2 and 3 can be very sensitive to contention since they involve reading several moderately sized files for compaction. Methods 2 and 3 should experience the same compaction performance but, likely due to increased contention, method 2 is significantly slower. For the local service mode, methods 0 and 1 actually perform best. The total database sizes are small enough that it actually takes longer to remove already checkpointed metadata from the database than it does to re-checkpoint it (method 0) or search for and output only the new metadata (method 1). Here again we see that the cost of compacting the databases far outweighs the checkpointing cost for methods 2 and 3, and makes these the worst performing option.

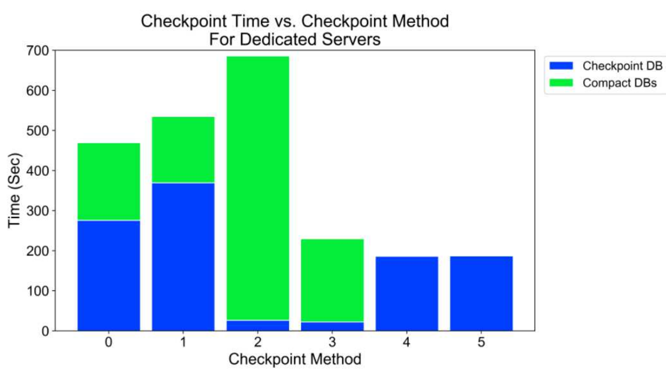

Figure 12

5.4.6 Transactions. Figure 14 demonstrates that both of our newly implemnted transaction methods, using a different database per transaction and using a lower-overhead method that limits the flexibility of transactions, result in significantly faster performance. Using a different database per transaction dramatically reduces write times since writes are performed to an empty or nearly empty database, but actually increases transaction management time since, upon committing a transaction, all of the writes must be copied to the "committed" database. We would, however, still expect this transaction method to scale better than the D^2T method since the

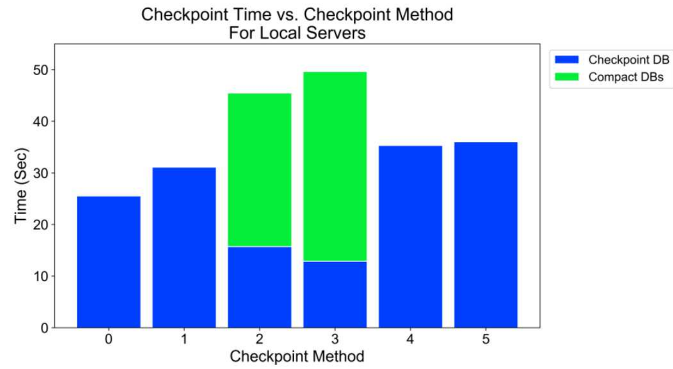

Figure 13

commit method is O(transaction size) rather O(log(total metadata size)). Note that it is O(log(total metadata size)) since we are using an index to perform the table scan. As expected, the fastest method is the one with the most limited functionality. By limiting the ability to have concurrent transactions, flexibility over what is removed by a rollback, and when reads can be performed, we obtain ample speedups. Writing is faster since we eliminate the need to write the transaction visibility status for each piece of metadata and the indices on this information, and the transaction management is much faster since we do not have to perform an (indexed) full table scan to commit a transaction.

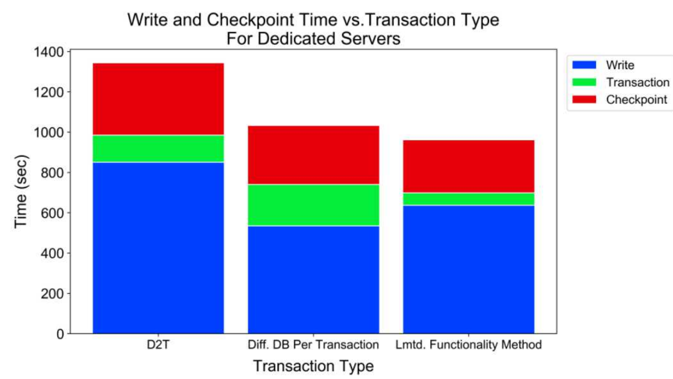

Figure 14

5.4.7 Indexing. Figures 15 and 16 demonstrate the performance impact of delaying the creation of indices until the end of the run. For the dedicated case, eliminating indices causes read time to jump from 2.61 seconds to 1317.75 seconds. However, we can also see that writing takes significantly less time, since each write no longer has to update the various indices. We can also see that the checkpointing time is slightly reduced (since the total database size is reduced), and very little compaction time is required (since the database was approximately 50% smaller, it was mostly able to fit in memory). However, indexing does require a significant amount of time at the end. This would, however, be done independently of the compute processes, and thus by trading indexing time on the server for faster writes with the compute processes, this could result in substantial savings in compute hours, if there are no or minimal reads required. For the local case, the picture is quite different. Writes take similar times and reading takes less than 2X longer for the non-indexed case (13.58 vs. 7.69). Overall, the relatively small size of the database changes most of the calculus, making delaying indices more feasible for cases with intervening reads and for cases that require more frequent checkpointing.

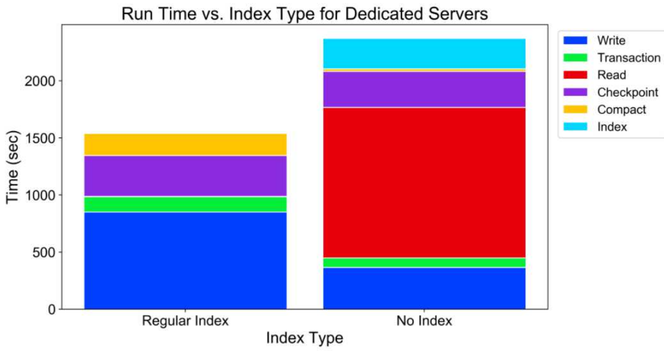

Figure 15

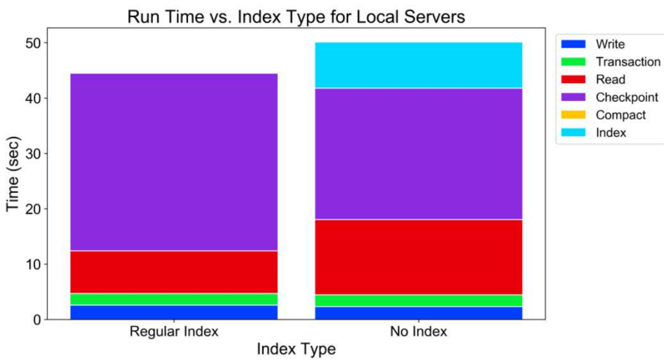

Figure 16

5.4.8 Message Bundling and Synchronicity. Two design features for which we do not present evaluation are message bundling and synchronicity. This is partially due to space constraints, and also due to the fact that their effect on performance will be very context dependent. The performance benefits gained by message bundling will be largely dependent on the number of messages bundled together, which processes need a response, the networking topology, and process distribution within this network. The performance benefits of using asynchronous operations will be dependent on how long compute phases last (assuming traditional cycles of compute and then output), and how quickly users need a response and from which operations.

6 RELATED WORK

While many tools provide some form of metadata management, far fewer offer support for the kind of descriptive, custom metadata that SystemX is designed to support. Of the tools that do support descriptive metadata, most suffer from a significant limitation that differentiates them from SystemX. The most common limitations are tools offering only file-level metadata, storing the metadata using key-value stores, being domain dependent, or lacking support for extensible, user-defined metadata.

File-Level Metadata. Many projects have focused on metadata management at the file level, meaning metadata that applies to an entire file. This includes tools that manage basic file system metadata such as GUFI [5], which is part of MarFS [7] from Los Alamos National Lab. This also includes tools that allow users to add limited custom annotations to entire files such as TagIt [24], ExpressQuery [14], Starfish [2], and POSIX extended attributes. While file-level metadata can be useful, it does not offer the level of granularity needed for either the petascale or exascale era. With individual files already exceeding several petabytes, scientists need access to finer-grained metadata to be able to limit their reading scope to data of interest and thereby accelerate analysis.

Key-Value Stores. Many solutions offer metadata management by allowing users to create key-value attributes. This includes the most popular I/O systems used by scientific simulations: ADIOS [20], HDF5 [11], netCDF-4 [23], and PnetCDF [17]. This also includes many tools that support only file-level metadata such as Starfish [2] and POSIX extended attributes, tools such as MIQS [30], which provides indexing of key-value attributes, and SoMeta, which offers a robust range of key-based metadata queries. However, key-value stores cannot efficiently support the wide range of queries needed by scientific users. Key-value stores suffer from two limitations. First, while they offer good performance when retrieving a value associated with an entire key, for all other searches they must resort to linear searches of all stored metadata. This poor performance makes key-value stores a poor fit for scientific users who need to be able to perform searches based on many different potential values such as run, timestep, variable, spatial area, and value. Second, for searches that do not involve the entire key, string matching must be used for each key in the store to determine if it is a match.

Domain Specific Solutions. A lot of work has been done to develop domain- and application-specific tools to aid with metadata management. Many of these tools use database backends to offer a wider range of querying capabilities. Examples include the Catalog Archive Server (CAS) [25] for the Sloan Digital Sky Survey (SDSS), ATLAS [4] for the Large Hadron Collider, the Atmospheric Data Discovery System (ADDS) [22], the Biomedical Image Metadata Manager (BIMM) [15], the JGI Archive and Metadata Organizer (JAMO) for genomics [3], and the SPOT Suite for advanced light sources [26]. While these systems have their merits, they offer entirely domain-specific solutions, which do not offer the generality or flexibility offered by SystemX. They also do not allow for extensible, user-defined attributes like SystemX since they are designed to capture particular, predefined kinds of features and to generate standardized metadata catalogs.

Limited Extensible, User-Defined Metadata. Many systems offer limited support for extensible, user-defined metadata, and instead focus on metadata that is automatically collected or which has little flexibility. This includes the domain specific solutions listed above and the Scientific Data Manager (SDM). The SDM uses a database to store metadata about the physical locations of data objects and abstracts away low-level storage details from the user. It also offers very limited basic attribute capabilities. These kinds of systems are designed to capture predefined categories of metadata and lack the flexibility and range of functionality needed to support users across the scientific domains.

7 FUTURE WORK

Future work will focus on expanding SystemX'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 SystemX can better serve applications with different data models, such as genomics applications. We also want to look further at improving usability to provide support for users with varying levels of comfort with programming. Finally, we hope to explore more fully how a metadata management service like SystemX can be integrated with storage systems to make better decisions about prefetching, tiering, and striping, and to better support the full data life-cycle.

ACKNOWLEDGEMENTS

[blinded for review]

REFERENCES

[1] [n.d.]. . http://www.sqlite.org/
[2] 2017. *Starfish*. https://storageconference.us/2017/Presentations/Farmer.pdf
[3] 2018. *JAMO - JGIArchive and Metadata Organizer*. https://storageconference.us/2018/Presentations/Beecroft.pdf
[4] Solveig Albrand, Thomas Doherty, Jerome Fulachier, and Fabian Lambert. 2008. The ATLAS metadata interface. In *Journal of Physics: Conference Series*, Vol. 119. IOP Publishing, 072003.
[5] David John Bonnie. 2018. *GUFI Overview*. Technical Report. Los Alamos National Lab.(LANL), Los Alamos, NM (United States).
[6] Suren Byna, Mohamad Chaarawi, Quincey Koziol, John Mainzer, and Frank Willmore. 2017. Tuning HDF5 subfiling performance on parallel file systems. (2017).
[7] Hsing-bung (HB) Chen, Gary Grider, and David Montoya. 2017. An Early Functional and Performance Experiment of the MarFS Hybrid Storage EcoSystem. 59–66. https://doi.org/10.1109/IC2E.2017.22
[8] Jacqueline H Chen, Alok Choudhary, Bronis De Supinski, Matthew DeVries, Evatt R Hawkes, Scott Klasky, Wei-Keng Liao, Kwan-Liu Ma, John Mellor-Crummey, Norbert Podhorszki, et al. 2009. Terascale direct numerical simulations of turbulent combustion using S3D. *Computational Science & Discovery* 2, 1 (2009), 015001.
[9] Jai Dayal and Jay Lofstead. [n.d.]. . https://github.com/gflofst/d2t
[10] Peter Dewdney, Peter Hall, R Schillizzi, and J Lazio. 2009. The square kilometre array. *Proceedings of the Institute of Electrical and Electronics Engineers IEEE* 97, 8 (2009), 1482–1496.
[11] Mike Folk, Gerd Heber, Quincey Koziol, Elena Pourmal, and Dana Robinson. 2011. An overview of the HDF5 technology suite and its applications. In *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*. ACM, 36–47.
[12] T. Ryan Gregory, James A. Nicol, Heidi Tamm, Bellis Kullman, Kaur Kullman, Ilia J. Leitch, Brian G. Murray, Donald F. Kapraun, Johann Greilhuber, and Michael D. Bennett. 2006. Eukaryotic genome size databases. *Nucleic Acids Research* 35, suppl_1 (11 2006), D332–D338. https://doi.org/10.1093/nar/gkl828 arXiv:http://oup.prod.sis.lan/nar/article-pdf/35/suppl_1/D332/3835543/gkl828.pdf
[13] Zeljko Ivezic, JA Tyson, B Abel, E Acosta, R Allsman, Y AlSayyad, SF Anderson, J Andrew, R Angel, G Angeli, et al. 2008. LSST: from science drivers to reference design and anticipated data products. *arXiv preprint arXiv:0805.2366* (2008).
[14] Charles Johnson, Kimberly Keeton, Charles B Morrey III, Craig AN Soules, Alistair C Veitch, Stephen Bacon, Oskar Batuner, Marcelo Condotta, Hamilton Coutinho, Patrick J Doyle, et al. 2014. From research to practice: experiences engineering a production metadata database for a scale out file system.. In *FAST*. 191–198.
[15] Daniel Korenblum, Daniel Rubin, Sandy Napel, Cesar Rodriguez, and Chris Beaulieu. 2011. Managing biomedical image metadata for search and retrieval of similar images. *Journal of digital imaging* 24, 4 (2011), 739–748.
[16] S Ku, CS Chang, and PH Diamond. 2009. 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* 49, 11 (2009), 115021.
[17] Jianwei Li, Wei keng Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, and M. Zingale. 2003. Parallel netCDF: A High-Performance Scientific I/O Interface. In *Supercomputing, 2003 ACM/IEEE Conference*. 39–39. https://doi.org/10.1109/SC.2003.10053
[18] Jay Lofstead, Jai Dayal, Karsten Schwan, and Ron Oldfield. 2012. D2t: Doubly distributed transactions for high performance and distributed computing. In *Cluster Computing (CLUSTER), 2012 IEEE International Conference on*. IEEE, 90–98.
[19] Jay Lofstead, Milo Polte, Garth Gibson, Scott Klasky, Karsten Schwan, Ron Oldfield, Matthew Wolf, and Qing Liu. 2011. Six degrees of scientific data: reading patterns for extreme scale science IO. In *Proceedings of the 20th international symposium on High performance distributed computing (HPDC '11)*. ACM, 49–60. http://doi.acm.org/10.1145/1996130.1996139
[20] Jay Lofstead, Fang Zheng, Scott Klasky, and Karsten Schwan. 2009. Adaptable, Metadata Rich IO Methods for Portable High Performance IO. In *In Proceedings of IPDPS'09, May 25-29, Rome, Italy*.
[21] Ron A. Oldfield, Patrick Widener, Arthur B. Maccabe, Lee Ward, and Todd Kordenbrock. 2006. Efficient Data-Movement for Lightweight I/O. In *Proceedings of the 2006 International Workshop on High Performance I/O Techniques and Deployment of Very Large Scale I/O Systems*. Barcelona, Spain.
[22] Sangmi Lee Pallickara, Shrideep Pallickara, and Milija Zupanski. 2012. Towards efficient data search and subsetting of large-scale atmospheric datasets. *Future Generation Computer Systems* 28, 1 (2012), 112–118.
[23] R Rew, E Hartnett, J Caron, et al. 2006. NetCDF-4: Software implementing an enhanced data model for the geosciences. In *22nd International Conference on Interactive Information Processing Systems for Meteorology, Oceanograph, and Hydrology*.
[24] Hyogi Sim, Youngjae Kim, Sudharshan S Vazhkudai, Geoffroy R Vallée, Seung-Hwan Lim, and Ali R Butt. 2017. 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, 5.
[25] Ani R Thakar, Alex Szalay, George Fekete, and Jim Gray. 2008. The catalog archive server database management system. *Computing in Science & Engineering* 10, 1 (2008).
[26] Gunter Dan et al. Tull Craig E., Essiari Abdelilah. 2013. *The SPOT Suite project*. http://spot.nersc.gov/
[27] Craig Ulmer, Shyamali Mukherjee, Gary Templet, Scott Levy, Jay Lofstead, Patrick Widener, Todd Kordenbrock, and Margaret Lawson. 2018. Faodel: Data Management for Next-Generation Application Workflows. In *Proceedings of Workshop on Infrastructure for Workflows and Application Composition (IWAC), 2018*.
[28] WX Wang, ZTWM Lin, WM Tang, WW Lee, S Ethier, JLV Lewandowski, G Rewoldt, TS Hahm, and J Manickam. 2006. Gyro-kinetic simulation of global turbulent transport properties in Tokamak experiments. *Physics of Plasmas* 13, 9 (2006), 092505.
[29] Sage A Weil, Scott A Brandt, Ethan L Miller, Darrell DE Long, and Carlos Maltzahn. 2006. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th symposium on Operating systems design and implementation*. USENIX Association, 307–320.
[30] Wei Zhang, Suren Byna, Houjun Tang, Brody Williams, and Yong Chen. 2019. MIQS: metadata indexing and querying service for self-describing file formats. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 5.