This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

SAND2019-4650C

A Vision for Managing Extreme-Scale Data Hoards

Jeremy Logan[†], Kshitij Mehta[*], Gerd Heber[‡],
Scott Klasky[*†], Tahsin Kurc[*§], Norbert Podhorszki[*], Patrick Widener[¶], Matthew Wolf[*]
[*]Oak Ridge National Laboratory, Oak Ridge TN, USA
[†]The University of Tennessee, Knoxville TN, USA
[‡]The HDF Group, Champaign IL, USA
[§]Stony Brook University, Stony Brook NY, USA
[¶]Sandia National Laboratories, Albuquerque NM, USA

ABSTRACT

Scientific data collections grow ever larger, both in terms of the size of individual data items and of the number and complexity of items. To use and manage them, it is important to directly address issues of robust and actionable provenance. We identify three key drivers as our focus: managing the size and complexity of metadata, lack of *a priori* information to match usage intents between publishers and consumers of data, and support for campaigns over collections of data driven by multi-disciplinary, collaborating teams. We introduce the Hoarde abstraction as an attempt to formalize a way of looking at collections of data to make them more tractable for later use. Hoarde leverages middleware and systems infrastructures for scientific and technical data management. Through the lens of a select group of challenging data usage scenarios, we discuss some of the aspects of implementation, usage, and forward portability of this new view on data management.

***Index Terms*—data provenance, reproducibility, metadata management, scientific data management**

I. INTRODUCTION

As scientific data sources, whether experimental, observational, or simulation, have continued to scale, managing the data life cycle of the primary and derived datasets and data elements (represented as files or data objects) has also grown to be a large problem. All practitioners develop some standards for how to organize their data in order to be able to answer questions like, "Which folder, among the hundreds that are part of this project, has the data that yielded the result in our most recent paper?" Tools used to address these questions tend to be coarse-grained (i.e., using *ls* and *grep* to find all the files from September of last year), as well as prone to error ("My input parameter file for the October data was the good one from the August runs... or was it the July runs?").

Here we present an abstraction for collection management that we call Hoarde; it is a vehicle for taking the large data pool (your hoard) and turning it into a loosely regimented set of agents that can get you answers (your horde). Hoarde is an attempt to formalize a way of looking at collections of data to make them more tractable for later use. In other words, it seeks to take a vast collection of piled data (a hoard) and convert it into something that can lead to actionable advancements (a horde). Hence the name – hoard+horde = Hoarde. More concretely, a Hoarde is a model for representing campaign metadata that is light-impact, descriptive, highly flexible, uses existing tools, supports large volumes of data, and is built upon incorporation of self-describing data.

In particular, however, Hoarde is addressing a particular rising set of scientific data hoards that are composed of collections of self-describing data. Self-describing data formats like HDF5, ADIOS, XML, NetCDF, and so on are an integral part of many existing simulation I/O infrastructures, and their impact has grown over the years. The advantage of self-describing data formats is that they provide a vehicle for later users (potentially the same scientists) to come back and not only access the raw binary values of the data elements (variables) but also query and interpret based on the local context of those variables. For example, I may call pressure "Pr" in my current code, but I recognize the variable named "Pressure" as being the same thing.

We build upon a foundation of projects that have explored aspects of this problem over the years. Ranging from explorations of novel approaches for light-weight self-describing data formats [1], [2], [3] to extending data sources with embedded visualization [4] or performance metadata [5], we have demonstrated a consistent value in maintaining some of the context provenance of individual data sets as they are generated. We also have shown the utility of later repurposing such enhanced data for tasks like automatically generating I/O performance benchmark codes [6], interfacing to pre-packaged visualization routines [7], or using it as a building block for future designs of I/O systems [8]. As valuable as self-describing techniques have been for maintaining a connection for individual data items and files, there still remains a problem of how to deal with large collections (data lakes, data hoards, piles of files, etc.). Experiences with provenance systems [9], [10] has shown that there is some redundancy between what must be manually input by a user for the provenance and what is available in the self-describing format.

Leveraging this track record of using techniques for real application scenarios as well as a wealth of related work (§V), there are three key drivers we have identified as our focus: Managing the size and complexity of metadata; Lack of *a priori* information about what indices or provenance markups will be needed; and Support for campaigns over collections of data driven by multi-disciplinary, collaborating teams. 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.

context for these is more fully developed in the analysis of some of the use cases detailed in § II-D.

As our approach, we have targeted finding a minimal intrusion, without complex runtime and setup dependencies, that can help support the relevant uses. We are exploring how to do something with a lighter impact than traditional provenance systems based on the combination of self-describing data and the context clues that come from data being in a collection. Much like reflection or introspection can be used in a programming model to enable interesting new capabilities, these self-describing collections can be used to enable new data-centric services.

A richer description of the Hoarde abstraction is in § III; a key insight is that we want minimal user impact that enables key operations that align with use case scenarios similar to those described in § II. To this end, we have focused on operations of key membership, forward provenance, and backward provenance, rather than generic, full provenance views. These operations include, "is this image from my paper in this archive" (compare: *cmp*), "What collection items were part of generating this item?" (list: *ls*), "What were the differences between the constituents for generating these two items" (difference: *diff*). The comparisons to traditional file system operators are not exact; for example, the return value of an "ls"-like operator on a collection member is an annotated provenance graph, rather than a simple name and modification timestamp. However, it is not a complete surprise that there are many similar analogies for the basic functionality from the filesystem community, as that is how many end users have chosen to create their own ad hoc collection management systems. In our work, we have built an infrastructure to understand some of the details of how the content and presentation would be most useful for these sorts of queries over collections and to explore the performance and metadata structure implications of this set of restrictions, which we discuss in depth in § IV.

We present a few results from the infrastructure, giving early definition to where we think further work is needed. For example, longitudinal studies of large data collections are an important scenario in data-rich environments that our tools should support. Collection-tracking and automation around self-describing data is also useful beyond the scenarios described in this work. Such tools could be layered on top of many measurement-driven datasets (e.g., those generated from Internet of Things, scientific sensors, and performance quantification studies).

II. Motivating Examples

For this work, we focus on a different subset of large data hoards than the cloud-scale data lakes, data warehouses, etc. We build from experience with large-scale scientific and technical data sets, but we also are informed by how large, extremely heterogeneous data collections (like those from academic libraries or federal process review) have similar constraints. With the scenarios below, we wanted to capture a host of such usages. This scenario in II-C specifically is a useful challenge to our basic assumptions, since much of the data may have been originally written without a self-describing format, but light-weight ways for bridging this divide will hopefully emerge from its exploration. With this basis of scope, we then discuss the commonalities that the Hoarde abstraction seeks to address.

A. Performance Optimization at the Exascale

Cooperative design (Co-design) centers that foster a collaborative approach between software ecosystems, hardware technologies, and computational science applications form an essential part of the Exascale Computing Project (ECP) [11], [12]. One such co-design center is CODAR (Co-Design Center for Online Data Analysis and Reduction at the Exascale) [13] that aims to study performance tradeoffs for offline versus online analyses of data for different classes of applications.

Such co-design studies involve coupling multiple applications using an I/O middleware such as ADIOS [1]. Experiments are performed to study the impact of various compression and reduction algorithms under different workflow configurations. Fig 1 shows a real-world example of the high-fidelity whole device modeling of magnetically confined fusion plasmas [14], [15]. Here, XGC and GENE are simulations that are tightly coupled through the ADIOS middleware. Different output variables undergo different transformations before some set of results is written to long-term storage.

Co-design studies typically consist of a large number of experiments, run by multiple users, on multiple supercomputers, and across different applications in various domains. A wealth of performance information is needed to understand the tradeoffs and impact of different choices. Simulation inputs, outputs, and reduction/analysis results as well as performance data are captured and stored in self-describing formats.

CODAR developed a tool, called Cheetah, to generate the campaigns of experiments needed to conduct such parametric studies. A Cheetah campaign directory consists of separate sub-directories for each experiment, along with extensive metadata about the workflow, its orchestration, and simulation and performance output data. That is, each co-design experiment produces multiple intermediate and final data products. An interesting aspect of co-design data is that some data itself forms metadata; metadata does not necessarily exist as a separate entity. For example, consider a query where the user wants to know the fidelity of a compression method as compared to a base case. Here, the base case forms the metadata; however, it exists as a set of experiments in the campaign and is not tagged, marked, or stored any differently from other data. Traditional metadata and provenance systems fall short here as most of them require metadata to be marked and stored in a different way from other data.

As a detailed analysis of a campaign is performed, the following features are highly desired.

- Being able to establish the detailed lineage for various data products in co-design experiments is important for multiple reasons. First, it greatly aids in making data and processes reproducible. Second, it makes debugging

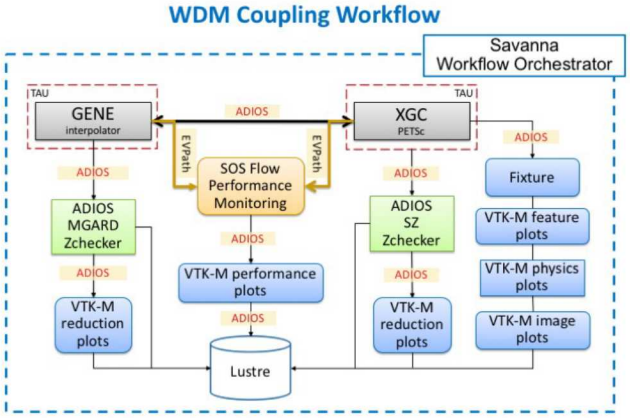


Fig. 1. The *Whole Device Modeling* coupling workflow is a typical co-design experiment in which two fusion simulations - XGC and GENE, are tightly coupled together using ADIOS. Multiple diagnostics and analysis executables are coupled with the running simulations. Co-design experiments that optimize this workflow and study various tradeoffs involve running a large number of experiments, thereby generating a large volume of campaign data.

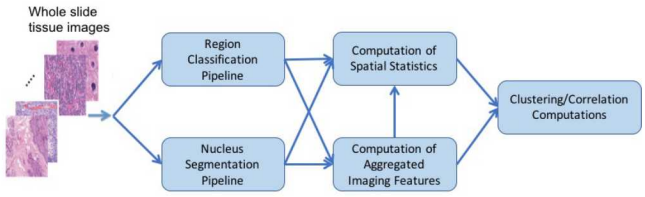


Fig. 2. *An example of analysis methods and workflows to carry correlative analysis using whole slide tissue images. Tissue images are processed through groups of analysis pipelines to extract object-level and region-level imaging features. These features are then processed through another set of pipelines to compute statistics and summaries. Finally, the summary data are analyzed to generate patient-level clustering and correlation data to look for relationships between imagging features and genomics or clinical data. A study like this will execute multiple variations of individual analysis pipelines, generate a a large number of derived datasets, and create a complex graph of analysis operations and datasets. Metadata and provenance information has to be captured, managed and indexed for the research team to debug the whole process and produce high quality results for publication.*

issues in complex workflows easier. Third, it allows making a comparative study of workflows. Questions such as "what was different between these experiments that led to this different result?" can be answered effectively.

- Querying large amounts of campaign data for information is very challenging, as the structure of the data is not known a priori. This prohibits the use of database systems that rely of building relational schemas based on well structured data. Complex queries such as "what operation in the complex workflow led to this intermediate data item?", "what compression scheme leads to at least 50% reduction in data size?" are highly desired. The best way to index campaign data automatically is a challenging question.
- Maintaining historical campaign data facilitates development of different machine learning algorithms focused on optimizing workflows, and also for effective dissemination of information to the science community.

B. Analysis of Whole Slide Tissue Images

High resolution images of diseased and normal tissue specimens enable quantitative studies of disease state at the subcellular scales [16], [17], [18], [19], [20]. As technology for scanning whole slide tissue specimens rapidly improves, the variety and sizes of tissue image datasets and the complexity of image analyses increase. In most cases, an imaging project will employ multiple analysis pipelines and improve analysis results in an iterative process. Consider a study investigating correlations between tumor morphology and cancer sub-types or clinical outcome data (see Figure 2). One group of analysis pipelines in this study processes whole slide tissue images (WSIs) to classify regions and extract patterns of lymphocytes. Another set of pipelines segment nuclei in the images and compute size, shape and texture features for each segmented nucleus. A third group of analysis pipelines computes spatial statistics and aggregate features at the image- and patient-level from lymphocyte patterns and nuclear segmentation results. The last set of analysis pipelines carries out clustering and association operations on spatial statistics and aggregated features to look for correlations between imaging features and cancer sub-types. Each of these pipelines may be composed of multiple methods. A nucleus segmentation algorithm, for example, may consist of a series of operations, including color normalization, color de-convolution, nucleus detection, distance transformation, clustering, and mean shift analysis.

In this study, the research team will carry out analyses multiple times in order to improve the analysis results. In this process, the team may change the algorithm parameters or the algorithm stages (e.g., they may use a different clustering method), or even use new analysis pipelines. Different analysis runs may be carried out by different members of the research team. In addition to this metadata, the team has to capture and manage metadata about a large volume of primary and derived data. In medical imaging domain, DICOM and its various extensions [21], [22], [23], [24] as well as vendor specific formats provide domain specific self-describing data to capture metadata about images and analysis results. These formats, however, do not necessarily store provenance information. Moreover, WSIs are high resolution images and can contain tens of billions of pixels. WSIs are often partitioned into patches for analysis with region classification algorithms. Nuclear segmentation analyses may segment millions of nuclei in an image. A single analysis run on a dataset with thousands of images can generate hundreds of thousands of patches, segmentation masks and hundreds of millions of segmented nuclei. As a result, the iterative refinement process can easily create a large number of analysis runs, a large volume of derived datasets (analysis results), and a complex, evolving graph of data analysis operations and datasets.

C. Supporting legacy collections of ad-hoc heterogenous data

Some classes of applications involve long-running efforts that accumulate vast stores of different but equally important

data over time as artifacts of design, testing, and production. Design documents include reports, schematics, spreadsheets, emails, and other notes; these are produced by and manipulated with both widely available commercial productivity software (such as Microsoft Office) and by special-purpose software with proprietary storage formats. Testing data adds to this large amounts of numerical results, test descriptions, and parameter sets. Production data adds another type of data store to the problem, as database management software is frequently used to maintain inventory information. When problems are detected during testing or production use, answering the questions that lead to root causes requires a holistic look back at a large and interrelated data space: What testing regime was used for the widget in question? Was its design valid? How many of these widgets are in production? Data that accumulates over years becomes challenging to manage and query. Techniques for organizing different heterogeneous data in the right ways can shorten the iteration time required for such investigations.

These kinds of issues arise in the context of high-consequence continual experimental activities, such as those carried out as part of the National Nuclear Security Administration (NNSA) (a part of the US Department of Energy) Stockpile Stewardship [25] program. Since the early 1990s, the US has unilaterally refrained from testing nuclear weapons through actual explosive tests. Computer simulations are instead used to predict the reliability, safety, and performance of weapons and weapon components. Models of the various physical phenomena involved are regularly validated against historical data collected during previous underground testing, and are constantly reviewed, updated, and improved as new data becomes available from laboratory examinations, material properties experiments, and other evolving simulation results. As the results of the Stockpile Stewardship program are regularly reported to the US Congress, the organization of this data environment must satisfy rigorous audit and verification requirements.

Another example of such large, increasingly voluminous, heterogeneous data comes from academic libraries that are increasingly being tasked with campaign-scale data management for experimental research across multiple research disciplines. In this capacity, they are seeking to provide not only basic hardware and software solutions for data storage, but also advanced features and capabilities requested or required by faculty, researchers, and other stakeholders[1]. Notable among these capabilities is the ability to flexibly reconstruct previous experimental results on demand. The cross-disciplinary service orientation of academic libraries frequently means that they cannot dictate data storage formats, archival strategies, and "build-vs-buy"-style policies about storing data versus regenerating it. More often, the opposite is true: librarians have to accommodate researchers whose experimental data collection practices and ideas on appropriate data management can differ significantly. Issues which must be addressed here include: which data should be stored, at what availability and cost? When does it make more sense to store/tag a data generator rather than the data itself? How can new workflows incorporate data previously stored in a library's data management system in order to generate new derived data, and how should those associations themselves be recorded?

[1]At American universities, these can include Federal, state and local government agencies, private funding organizations, and community interest organizations.

D. Discussion

An overarching commonality amongst all the use cases described above is that they have strong requirements for managing provenance and metadata; however, no single provenance system meets the requirements of all the use cases. There is no 'one-size-fits-all' solution, but it would be highly beneficial if there were a high-level abstraction that could provide a generic solution, or which could be used as a baseline to design an ad-hoc solution. We identify three important features that are common across the use cases that influence the design of an abstraction such as Hoarde.

First, all use cases described involve managing large hoards of complex, heterogeneous data sets and annotations. For co-design experiments, this involves a combination of simulation data, analysis data, application and workflow provenance, workflow orchestration information, and performance information, stored across self-describing binary files, *json* documents, text files etc. For the image analysis use case, analyses performed over large amounts of tissue images generates a large volume of intermediate data products with important metadata that describes the workflow. The collections of data that need to be handled by the NNSA include documents that describe metadata about a variety of processes, accumulated over a period of many years. In all of these use cases, experiments, analyses and data capture processes can rapidly evolve. In the image analysis use case, for example, the structure of the analysis workflows and types of derived datasets can quickly change as the team seeks to iteratively do quality assessment and improve analysis results. This dynamic, rapid evolving nature of data collection and analysis necessitates flexible absractions and frameworks that have minimal impact on a research team's workflow and that can capture metadata in a self-describing format which can later be parsed, indexed, and managed.

Secondly, the distinction between data and metadata can be blurry, which is a hindrance for using traditional provenance systems. As described for co-design experiments, data or experiments that form baseline runs are considered to be metadata for later analysis and comparison across experiments. This classifies some data items as more reusable and add more weight to them in the hoard, and also has a strong effect on the quality of service requirements of an application. For example, what if a query does not return a data product that should have been returned if the data product were classified differently? As machine learning algorithms are expected to play a strong role in the characterization of complex data in a large hoard, this is an important consideration. So, how should Hoarde automatically associate data with metadata to capture both data and processes?

Thirdly, collection-tracking and (automated) metadata capture and management around self-describing formats provides an effective means of packaging and disseminating not only data but also relevant metadata context. In addition to being vital for reproducibility and dissemination to the scientific community, this is critical in multi-disciplinary and multi-institutional research where team work is not necessarily closely coupled and synchronized (e.g., the image analysis use case).

Finally, from a functional perspective, building and maintaining the tooling for any automated extraction is very challenging. Although one could implement the necessary features with a sufficiently complicated database infrastructure and clever schema, we want to look at what can be achieved with the most minimal extension/addition that still supports these requirements. We take these insights in order to propose our new abstraction for a collection of metadata-rich items, as you will see in the following section.

III. The Hoarde Abstraction

We envision Hoarde to be both an organizational convention around a dataset and its metadata, as well as the software mechanisms which support and leverage those conventions. These pieces will work together to address the common features summarized in § II-D. To characterize the Hoarde abstraction, we approach it from several perspectives in the remainder of this section.

A. Purpose / Objective

The main purpose of Hoarde is to support the management of campaign life cycles in HPC environments. Hoarde's descriptive approach attempts to create a useful metadata abstraction for managing data related to an HPC campaign, which consists of all data related to a set of workflow executions performed over a number of weeks or months. Based on our experiences with the science cases described in Section II and other science applications, we believe we have identified a critical set of behaviors and functionality that a metadata management system would need to support. This functionality includes provenance for reproducibility, dataflow debugging and execution support, as well as support for rich queries involving combinations of data and metadata. Whatever the subject, the underlying campaigns are driven by inquiry, which typically includes false starts, dead ends, and often unpredictable dynamics. Hoarde is an attempt to have a *stabilizing* effect on campaign data life cycles much like the scientific method and judicious applications of logic have on the process of scientific discovery.

B. Data

Self-describing data formats such as ADIOS, HDF5, and NetCDF already play a significant role in managing scientific data collections, particularly those that form coherent campaigns or chains of experiments. The goal of Hoarde is to be a minimal extension that will enhance those community practices. Although there is some similarity to existing provenance systems that depend on external databases and infrastructures to maintain the connections and context, we focus instead on the use of self-describing data formats which can support embedding metadata alongside the data.

It is important to note that for the vision we lay out below that the contents we envision in a Hoarde are more than just the immediate binary data files associated with the experimental data. It is important to include a complete repository within the campaign archive: source code for executables, python analysis scripts, output from std.out of the run, input files, and so on, as well as image files, analysis results, and other scientific output files. It is this richer context of the data setting within the data hoard that enables the inference of a set of provenance and connectedness properties.

With that in mind, the main relationship between datasets in a Hoarde is a simple derivation relation. A *source dataset* refers to a dataset that was not produced by a campaign but rather introduced as an input to the campaign. This is in contrast to a *derived dataset*, which is produced as a result of an *operation* on one or more inputs. We also define *metadata* broadly as any information added to a dataset to provide additional context, leaving the user free to establish connections to any of the relevant contexts.

A Hoarde consists of datasets and supports certain operations on the metadata of those datasets. By 'metadata' we mean any information which documents the *derivation relationship* between datasets: As shown in Fig. 3, some datasets are the result of executing certain *commands*, of applying *processing steps* to other datasets. (See similar concepts in [26] or [27].) A command can be a complex workflow or a single processing step.

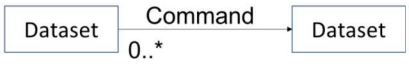


Fig. 3. A dataset is produced from an operation on zero or more existing datasets.

This relationship leads to a simple categorization of datasets, which are illustrated in Fig. 4. A dataset which is not derived from other datasets (in the same Hoarde) is called a *source* dataset. All non-source datasets in a Hoarde are referred to as *derived* datasets. It is sometimes convenient to further divide derived datasets into *intermediate* and *product* datasets. Product datasets are derived datasets that have additional metadata, such as a goal, an intended audience, a release date, etc.

For example, performance studies described in section II-A involve generating a large number of experiments in a campaign. Each experiment consists of a set of simulations running concurrently and coupled together to produce a set of raw, binary output data. These simulations are controlled by input datasets and parameters described in parameter files. Application parameters, workflow orchestration options are maintained in separate files. Multiple analysis applications may be run over the simulation output (FFT calculation,

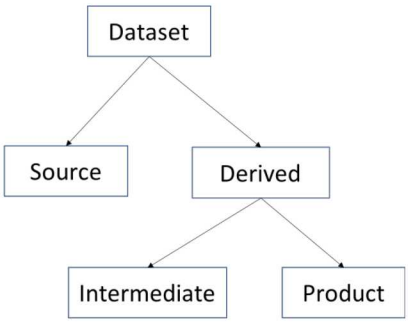


Fig. 4. Types of datasets

probability distribution function generation etc.) to further generate data products. These are followed by plotting scripts written in languages such as Python and R to generate a set of images. In a Hoarde, the input data files form the source datasets, whereas the simulation outputs form the intermediate datasets. They further lead to another set of intermediate datasets (output of the FFT calculations), which then lead to the final data products - the images. The parameter files that can be read and ingested by Hoarde form the core metadata, and a combination of parameters with applications that are run on intermediate datasets form the enactments or commands.

Users should be able to store arbitrary metadata as needed for an application, but there is a minimal collection of information we would like to capture for every dataset. Generally this consists of the information needed to allow the user to recreate the steps that produced the dataset in question. For a particular dataset, this might include details of the program that produced it (including, for instance, a link to a particular commit in a source repository), command line arguments, input files, etc. In general, results are not always produced from a single workflow specification processed by a workflow engine. For non-trivial workflows, we would expect a final product to have a tree of metadata leading back through multiple program executions to leaf nodes consisting of external source datasets. Some or all of the data may be produced by manual steps, or by multiple workflow descriptors, so some care is required to insure that enough information is captured to describe all of the necessary steps to reproduce a particular data set.

A longer term goal is to have an automated mechanism that could, from the stored metadata, reproduce the workflow in question without user intervention. However, this is a much more challenging goal, as it would require a full accounting of the system software on the machine, and a sufficiently adroit mechanism to build an arbitrary software stack on demand.

There are a number of options for organizing metadata, and a given choice will have significant impact on system performance. Fig. 5 illustrates three possible choices. In Fig. 5a, all campaign metadata is stored and maintained in a single location. Keeping metadata together simplifies maintenance of indices as new metadata is added to a Hoarde. However, scalability may become an issue as the campaign grows larger. Another strategy, shown in Fig. 5b, is to keep shadow metadata. In this scheme, separate metadata files are kept for each dataset and each operation. At the other end of the spectrum, Fig. 5c shows the use of self describing data files to store metadata alongside data. A full fledged implementation would potentially use a combination of all three of these strategies to juggle different concerns, switching between strategies to facilitate creation of query indices and support tightly-coupled metadata, while allowing arbitrary file formats.

The advantages and disadvantages of these organizational options become clearer as we begin to examine some common operations we expect Hoarde to perform. The insertion operation, for instance, is impacted by the choice of metadata organization. This is likely an insignificant cost for insertion of a single data object, but it becomes more of a concern when we consider ”bulk import” of existing project data, which may consist of large numbers of individual data files.

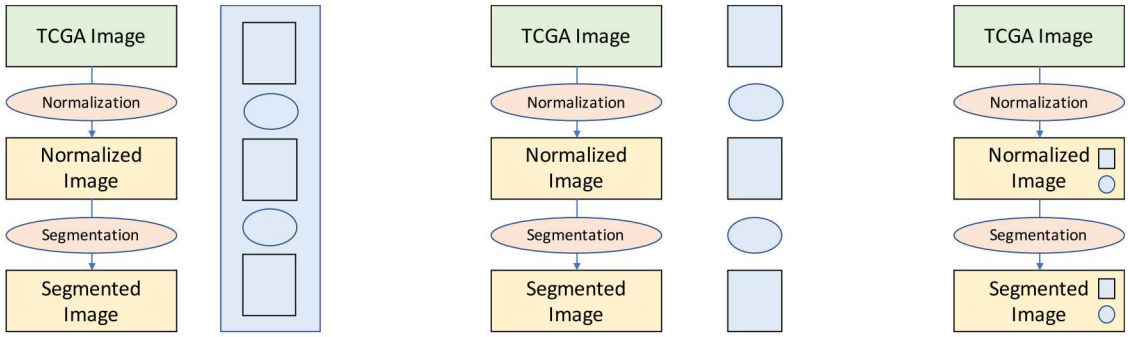
Another axis in the design space relates to how Hoarde controls access to data. Regardless of metadata organization, the system will rely on links between metadata and corresponding data. Unregulated, users would be able to move or delete files from their known locations, resulting in broken metadata links. How should a system like Hoarde handle this issue? One option would be to do nothing, allowing the user to move files and break links between metadata and associated data. At a minimum, a system like this ought to allow the user to update and repair these links. However, this is probably not an acceptable strategy as we begin to rely more on these systems for accountability and reproducibility, as there would be little to insure that metadata remains correct after such manipulations. At the other extreme, our system could strongly limit access to all data and metadata, requiring a user to access all data and metadata through a system API that hides data and metadata locations, and limits changes that can be made by users. Exploring this design space and understanding the many trade-offs is a part of the goal of our initial experiments described in the next section.

C. Behavior / Functionality

Hoarde behavior can be broken into three groups of functions to facilitate:

1) Data provenance capture for reproducibility
2) Data- and workflow debugging, and (re-)execution
3) Combined queries of data and metadata.

Provenance for reproducibility: Full reproducibility of workflows is a challenging goal, and one that is beyond the scope of this work. However, we believe that a key ingredient of this reproducibility is the availability of appropriate provenance metadata. For a particular data product, this metadata should include a tree of operations and intermediate data products that are part of the campaign and leading back to any source (external) datasets that the product relies upon. Metadata for each operation should include source code or version information, along with input parameters, dependency versions, system information, etc., that would allow the operation to be performed again in a new workflow designed to reproduce the original computations. This collection of



(a) *Single metadata*: All campaign metadata is encoded in a single metadata repository.

(b) *Shadow metadata*: Metadata for each pipeline element is kept in a separate file, with links that define the connections.

(c) *Embedded Metadata:* Self-describing data files store both data and metadata.

Fig. 5. Metadata Organization: Each figure illustrates a possible arrangement of metadata for a simple image processing pipeline. Green indicates external source data, operations on the data are shown in red, derived data files are in yellow, and blue indicates corresponding metadata.

provenance information can also be leveraged to provide correct attribution of the source datasets that contribute to a published data product.

Executing and debugging dataflows: Working with a campaign involving a large set of runs that are performed over a long period of time tends to become more difficult as time passes, as memory of which workflows were performed, and what options were used, and how codes were changed fades from memory. To assist with these types of campaigns, a metadata management tool should answer questions about what steps were involved in generating a particular data product, and how the provenance of similar data products differ. It is critical that such a tool provide clear and concise answers to these questions, rather than exhaustive textual output that requires intense scrutiny by a scientist managing the campaign.

Metadata Versioning. Consider a use case of trying to track the relationship of a particular jpeg graph in a publication to the data sets and intermediate results that generated it. In terms of the data set derivation relationship maintained in a Hoarde, a publication is a product. This begs the question where the product's provenance and the provenance of all the other sources and intermediate datasets are tracked and maintained. To make matters worse, it would be unrealistic to assume that there is only one chain of versions, one provenance trail.

Campaigns are driven by inquiry, and false starts and dead ends are encountered regularly on many a campaign trail. Furthermore, most campaigns are team (multi-user) efforts, and different team members will explore different avenues simultaneously, learn from each other, and re-use each others results. While there appears to be a strong versioning component to this picture, it is the fundamental *non-linearity of progress* that must be handled in some way. Further, with the periodic export of parts of the Hoarde collection for use by remote collaborators and the occasional reintegration of their results, this non-linearity becomes even more evident. On the one hand, it is crucial that items in a Hoarde and their derivation are immutable (no backsliding, history can't be destroyed). On the other hand, it must be possible to "weave" results from different lines of inquiry into new products and further inquiry. This capability requires support for functional data structures [28]: *"A functional data structure is essentially an immutable data structure: its values never change. ..., functional data structures* do *support operations like insertion or deletion, they are just not in-place. Instead these operations are handled by creating an entirely new updated structure."*

Rich queries of data and metadata: Since science applications are primarily concerned with advancing science, we would also like to be able to provide a range of query capabilities that support access to both data and metadata, as well as combinations of the two.

Given the varied needs of our science use cases, it is also important to consider how the Hoarde implementation will need to support several types of queries. A Hoarde instance, for example, should be able to answer queries like, "Which folder, among the hundreds that are part of this project, has the data that yielded the result in our most recent paper?" Simply using file system tools to answer queries like this tend to be coarse-grained (i.e., using *ls* and *grep* to find all the files from September of last year) and prone to error ("My input parameter file for the October data was the good one from the August runs... or was it the July runs?"). Much like the use of *grep* and *ls* in command line scripts enable functionality, albeit at a cost, the Hoarde abstraction is intended to define (and provide tools to define) useful sets of primitives that can be composed together to support simple and complex queries. We envision future systems providing pluggable query mechanisms that will enable science domain users to design efficient domain-specific queries that, once added to the system, can be accessed through the front-end Hoarde mechanism. In the meantime though, we are focusing on a family of general queries that relate to common metadata that will be available regardless of application domain. Table I provides a listing of the core Hoarde queries on data and metadata. Supporting these types of queries will require the ability to build custom indices on-demand. In a distributed setting, the cost of building such indices for distributed, self-

Command	Description
includes <dataset>	Returns (Boolean) if a dataset is part of a Hoarde
describe <dataset>	Returns a description of a dataset
ancestors <dataset>	Returns the ancestors of a dataset
descendants <dataset>	Returns the descendants of a dataset
sources	Returns all sources of a Hoarde
sources <dataset>	Returns the sources of a given dataset
products	Returns all products of a Hoarde
products <dataset>	Returns the products that a given dataset influences
cmd <dataset>	Returns the command (parameters) that produced a given (derived) dataset
list <dataset>	Lists the lineage of a dataset
find <predicate>	Finds all datasets that satisfy a certain predicate
diff <dataset1> <dataset2>	Compares the lineage of two datasets
union <dataset1> ... <datasetN>	Returns a derived dataset that represents the union of certain datasets

TABLE I
HOARDE QUERY COMMANDS.

describing metadata as compared to centralized metadata is a key consideration.

D. Interfaces and Measuring Performance

There are additional perspectives on Hoarde which should be mentioned but which are still part of open exploration.

The first one is the question of the form of delivery; in other words, in what form does Hoarde get used by an end consumer? Is it a framework? A set of services? A library? An interface? A collection of tools? At the core is a need to quantify the usability of the resulting implementation of the abstraction for the end user. Usability is notoriously difficult to measure, as it has many aspects that are highly subjective. However, taking an approach where we can explore over several related implementations to provide similar capabilities but in different formats for end users will likely be an important component of the future vision. In this view, a goal for the future of the Hoarde abstraction is a shared *lingua franca* for distributed data engineering. End users should be able to use whatever sort of interface would best fit the local idiom; there is no global best framework or mechanism.

The second one is the development of policies for measuring performance of a Hoarde implementation. We describe in § IV a particular implementation choice, but the core of the work as we see it is in developing the policies of this data engineering environment and not in the particular idioms of the mechanism to enable it. Specifically, a key component of these policy decisions will have to be a quantification of metrics to assess the quality of the information extracted from the collection as a function of the efficiency of delivery. Here the efficiency is not necessarily about raw performance, but it also includes quantifying how many times a user needs to iterate on a set of queries to extract their desired result.

IV. PROTOTYPE AND EVALUATION

In this section, we report on what we have learned from trying to apply the Hoarde abstraction to the co-design and image use cases described in Section II. We describe a proof-of-concept based on Emacs Org mode, batch and python scripting, and Git, as well as analyse its limitations.

Our target platform was OLCF Titan. Since both use cases had been well underway for some time, it was also clear that this was not a "green field" exercise. We had to make do with the tools (mostly standard UNIX tools and scripting language) available on Titan and the digital artifacts from campaigns in progress.

A. Applying Hoarde ideas to Exascale Co-Design

A prototype implementation for this abstraction was made using fundamental concepts provided in the git revision control system. We use the git branching and commits model to provide a sample implementation for the performance studies for exascale described in section II-A. Recall that these co-design studies are composed of campaign of experiments, with a variety of data products in each experiment. To manage this Hoarde, we associate each atomic data item in the Hoarde with a *Research Object* (RO). These are unique ids that identify objects in the Hoarde. We generate a unique id for the campaign using UUIDs, and append the experiment number to generate a unique ID for an experiment as a simple illustration of creating uniquely IDs for ROs. Experiment ROs are maintained in a separate directory, which contains a full set of metadata to describe and compose the experiment.

We utilize git SHA-1 hashes to generate IDs for datasets involved in the experiment. We use git commits and branches to form the association between the various datasets in the RO, where a branch represents the lineage of a RO, and the commit order identifies the evolution of the workflow. The top-level commit represents a source dataset, whereas the leaf commits form the final dataset products. Experiment run parameters are extracted from input files, and automatically ingested in the RO for the experiment and stored separately as metadata. A query that inquires for the lineage of an image file would then involve calculating the SHA-1 hash of the image file and looking for it in the Hoarde. A successful match would then look up the git commits and the associated branch to post the full lineage of run parameters and input datasets that lead to the generation of the image. The lineage could be a simulation *that* read a parameter file and produced output files, *on which* an FFT application was run to generate data, *which* was analyzed using a Python script to generate the final image in question.

B. Versioning and Queries

As we observed in the introduction to Section III, research studies can have false starts, dead ends, and are increasingly team efforts which may explore multiple avenues concurrently. This nature of research studies requires a means of handling versioning through a functional data structure. One of our favorite tools, Git [29], implements just such a data structure [28]. The appeal and practicality of using Git in some form for provenance tracking was demonstrated in [30]. While Git alone is not sufficient for Hoarde, it is an excellent

tool for exploring the implementation of some of the Hoarde operations.

1) Using Git for Prototyping: For our prototype, we chose to implement three operations:

- `includes`
- `diff`
- `cherry-pick`

With a Git command namesake, the third operation is the most straightforward. In the tissue image analysis, for example, it is common practice to combine results (datasets) from different analysis runs into a combined result set that is not itself the direct result of an image analysis run, but represents the best judgement of domain experts. Assuming that the granularity of the commits is such that it facilitates this kind of "cherry-picking", there is a direct mapping with the required Hoarde functionality

The `includes` operation tests the affiliation of a dataset / digital artifact with a Hoarde. Users who are familiar with Git internals [29] know that Git maintains an internal object database in which byte copies of all versions of a repository's files are stored. The files are named or (content-)addressed essentially by their SHA-1 hash. An important advantage of this approach is that an item can be identified even if it was moved or renamed in a (location-based) file system. The calculation of SHA-1 hashes can take a substantial amount of time if it needs to be performed over large collections as is the case for the tissue image analysis. For our test collection of ∼115,000 artifacts, using a single processor, this took about an hour to calculate the SHA-1 hashes (including directory traversals, etc.). Fortunately, this operation can be easily parallelized and does not present a real bottleneck. The lookup of SHA-1s can be made more effective, by training a Bloom filter to quickly rule out the association of an artifact with a Hoarde. The Bloom filter for our sample collection was just a few MB. Finally, we maintained a sorted list of SHA-1 hashes to perform a binary search, if the Bloom filter test was inconclusive.

Implementations of a `diff` operation are highly dependent on representations and the complexity of the derivation relationships. In the co-design application, a campaign instance was represented as a DataFrame, where columns represent input parameters or (output) metrics and rows represent individual experiments. Here, `diff` can be reduced to the comparison of (potentially sparse) vectors and DataFrames. Because of the (non-trivial) graph structure of the image analysis workflow, to compare two such graph graph instances is less straightforward. A practical approach to arrive at an intuitive difference is not to treat this as a general graph comparison problem, but to reduce it to a largest common subsequence (LCS) problem as implemented, for example, in noWorkflow[31].

2) Why Git alone is not enough: Git served us well to explore some of the ideas related to functional data structures and versioning. However, it is clear that this is not exactly for what it was made. The most obvious shortcoming is the by comparison limited number of objects (a few million?) Git can handle. With respect to the derivation relationship, it is unclear where the command(s) would fit. Candidates would be the commit message or one of the Org documents in the commit, but there are plenty of other issues. As we mentioned earlier, the smallest unit of change in Git is the commit. Unfortunately, there is no evidence to assume that the evolution of many campaigns has a "natural" commit granularity. Even if it exists, it might hardly be known when a campaign gets underway, and it might change.

Another set of concerns is the *semantic gap* between the primitives of a given tool or technology and the domain concepts of a campaign. It's a non-trivial task for campaign planners to map domain concepts (e.g., representations of fields, calibrations, invariants, protocols, data documentation, etc.) onto the primitives offered by self-describing file formats. Note that the difficulty is *not* in the direction from the domain science to the technical primitives. Reduction is easy! The problem is in the reverse direction, when domain concepts need to be recovered from complex patterns of technical primitives. A Git commit that is part of multiple branches has a different domain-level *context* in each branch of which it is a part (it is "transcluded"). How is that context to be recovered, and where is the protocol that makes that recovery possible documented? (When artifacts are reduced to SHA-1 hashes, a "phantom multi-presence" can happen when, unbeknownst to the committer(s), two artifacts happen to be bytewise identical. When implemented this way, some Hoarde operations are in trouble immediately because the context is ambiguous.) This problem is not specific to Git: If we'd chosen a (functional) graph database, the problem would have been very much the same. The only difference would be in the low-level primitives (nodes, edges, and properties instead of commits, branches, tags, etc.).

Rather than trying to eliminate ambiguity (probably futile), to embrace it and to make the semantic gap manageable is perhaps the biggest challenge for Hoarde.

V. Related Work

There is a large body of research in provenance and reproducible analysis workflows. Simmhan et al. [32] provide a survey of methods, representations, and tools for data provenance in scientific research. Moreau et al. [33] organized a provenance challenge to evaluate provenance systems in terms of provenance capture, management, query and expressiveness of provenance representations. The challenge identified a set of common queries against provenance data. These queries include those for tracking how a result item was generated, searching provenance data based on (key,value) metadata, and comparing provenance data for two runs. Moreau et al.[34] proposed in 2011 a provenance model specification to support exchange of provenance models, a common representation upon which to build provenance tools, represent provenance for any data whether it was generated through a computer algorithm or another mechanism, and rules for making inferences on provenance information. Gil et al. [35] describe an ontology for capturing metadata about scientific software. Oliveira et

al. [36] propose a framework for analysis of data provenance. The framework provides support for provenance data capture, storage and inference, graph analysis, and visualization. Gil et al. [37] describe a software metadata registry so that scientists can identify, understand, assess software of interest, and can execute and update the software. Miao et al. [38] present a provenance and metadata management framework for collaborative data science workflows. This framework is designed to capture version lineages of artifacts (data, scripts, and results), workflow provenance, context metadata about artifacts, and data provenance. A prototype of the framework is implemented using git and Neo4j. The prototype provides tools for collecting provenance and metadata and for querying the provenance information.

Murta and Pimentel et al. [31], [39] describe a tool that captures provenance information from python scripts. The tool can capture different types of provenance using software engineering techniques such as syntax tree analysis and profiling. It provides support for graph-based and query-based analysis of provenance information. McPhillips et al. [40] propose a tool that allows users to annotate scripts so computational modules and workflow are explicitly represented. The tool captures the workflow by analyzing the annotations and facilitates a workflow-like visualization of the scripts.

There also has been work on methods and systems for metadata management and versioning. Scott et al. [30] describe an architecture to manage key-value metadata for data elements in a revision control system. They store and manage metadata and data files in a git repository. They use a separate branch for metadata files, allowing decentralized edits to metadata without touching data files. They define a set of rules for commits, branching, and merging to capture and propagate changes in metadata. Prabhune et al. [41] present a metadata management system designed to handle heterogeneous metadata models using a NoSQL database and RDF triple store. The system implements a metadata schema registry, workflow provenance management using ProvONE provenance model, and mechanisms for metadata quality control, and handling of dynamic metadata entries and changes in metadata. Bhardwaj et al. [42], [43] present a platform that allows users to version, split/merge, search and difference collections of datasets and carry out collaborative data analysis. Their system incorporates methods to handle large volumes of datasets and dataset versions and query support for analyzing differences between dataset versions and search for dataset versions of interest. Maddox et al. [44] propose a relational database branching system, called Decibel, to support versioning of datasets for concurrent analysis, integration, processing, and curation across teams. Decibel implements a relational database storage system with version control capabilities. Chavan et al. [45] present a system for versioning and provenance management in studies where data are collected or generated from different sources and by different teams (or sub-teams). They propose a query language to enable queries on versioning and provenance information. Bhattacherjee and Deshpande [46] describe a system for storage and query of versions of a dataset. Their system implements optimizations to efficiently store and query a large collection of versions of documents/records in a distributed environment.

VI. Conclusion

Starting from a set of experiences with building and managing collections of self-describing data sets, we have motivated a different abstraction layer for collection and managing provenance and associated metadata. Moving from "piles of files" to a coherent collection access interface is intended to give users access to richer sets of provenance and context information, without the additional time investment associated with populating most existing provenance systems.

We have introduced the Hoarde abstraction as a way to envision constructing such a thin layer for dealing with large collections of data. Although our broad vision is still in its early stages, we have drawn a few key conclusions from our early work: provenance tracking by itself (and its associated system implementation) is not sufficient to answer all of the interesting questions that arise, and embedding metadata alongside data in self-describing formats offers a strong advantage for later reuse, validation, and accessibility of data.

Adding semantics to data collections is a useful step, but we want classes of semantics so that we aren't creating specialized, bespoke approaches for each data collection. This can be extended in many ways depending on whether the relevant focus is bringing existing data collections into an active form, or if it is capturing online and traditional workflow provenance for future reuse and categorization of intermediate results. In either case, turning the data hoard into a horde of opportunities for getting your questions answered lies at the core of exploiting the extreme scale data of the present and the future.

Acknowledgment

Without the continued support from the Department of Energy's Office of Advanced Scientific Computing Research, the projects upon which this future vision rests would not be possible. Additionally, support from the DOE computing facilities in Oak Ridge and NERSC, as well as the National Science Foundation, were also critical.

References

[1] Q. Liu, J. Logan, Y. Tian, H. Abbasi, N. Podhorszki, J. Y. Choi, S. Klasky, R. Tchoua, J. Lofstead, R. Oldfield, M. Parashar, N. Samatova, K. Schwan, A. Shoshani, M. Wolf, K. Wu, and W. Yu, "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, may 2014. [Online]. Available: http://doi.wiley.com/10.1002/cpe.3125

[2] G. Eisenhauer, M. Wolf, H. Abbasi, S. Klasky, and K. Schwan, "A type system for high performance communication and computation," in *2011 IEEE Seventh International Conference on e-Science Workshops*, Dec 2011, pp. 183–190.

[3] P. Widener, G. Eisenhauer, K. Schwan, and F. E. Bustamante, "Open metadata formats: Efficient xml-based communication for high performance computing," *Cluster Computing*, vol. 5, no. 3, pp. 315–324, Jul 2002. [Online]. Available: https://doi.org/10.1023/A:1015637623058

[4] R. Tchoua, J. Choi, S. Klasky, Q. Liu, J. Logan, K. Moreland, J. Mu, M. Parashar, N. Podhorszki, D. Pugmire, and M. Wolf, "Adios visualization schema: A first step towards improving interdisciplinary collaboration in high performance computing," in *2013 IEEE 9th International Conference on e-Science*, Oct 2013, pp. 27–34.

[5] C. Wood, S. Sane, D. Ellsworth, A. Gimenez, K. Huck, T. Gamblin, and A. Malony, "A scalable observation system for introspection and in situ analytics," in *2016 5th Workshop on Extreme-Scale Programming Tools (ESPT)*, Nov 2016, pp. 42–49.

[6] J. Logan, S. Klasky, J. Lofstead, H. Abbasi, S. Ethier, R. Grout, S.-H. Ku, Q. Liu, X. Ma, M. Parashar *et al.*, "Skel: generative software for producing skeletal i/o applications," in *e-Science Workshops (eScienceW), 2011 IEEE Seventh International Conference on*. IEEE, 2011, pp. 191–198.

[7] U. Ayachit, B. Whitlock, M. Wolf, B. Loring, B. Geveci, D. Lonie, and E. W. Bethel, "The sensei generic in situ interface," in *Proceedings of the 2Nd Workshop on In Situ Infrastructures for Enabling Extreme-scale Analysis and Visualization*, ser. ISAV '16. Piscataway, NJ, USA: IEEE Press, 2016, pp. 40–44. [Online]. Available: https://doi.org/10.1109/ISAV.2016.13

[8] S. Klasky, M. Wolf, M. Ainsworth, C. Atkins, J. Choi, G. Eisenhauer, B. Geveci, W. Godoy, M. Kim, J. Kress, T. Kurc, Q. Liu, J. Logan, A. B. Maccabe, K. Mehta, G. Ostrouchov, M. Parashar, N. Podhorszki, D. Pugmire, E. Suchyta, L. Wan, and R. Wang, "A view from ornl: Scientific data research opportunities in the big data age," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, July 2018, pp. 1357–1368.

[9] M. Pierre, M. Vouk, S. A. Klasky, R. B. Tchoua, and N. Podhorszki, "Tracking files using the kepler provenance framework," 1 2009.

[10] M. A. Vouk, I. Altintas, R. Barreto, J. Blondin, Z. Cheng, T. Critchlow, A. Khan, S. Klasky, J. Ligon, B. Ludaescher, P. A. Mouallem, S. Parker, N. Podhorszki, A. Shoshani, and C. Silva, "Automation of network-based scientific workflows," in *Grid-Based Problem Solving Environments*, P. W. Gaffney and J. C. T. Pool, Eds. Boston, MA: Springer US, 2007, pp. 35–61.

[11] "The exascale computing project - co-design is key," https://www.exascaleproject.org/co-design-is-key-to-ecps-holistic-approach-to-capable-exascale-computing/.

[12] A. Almgren, P. DeMar, J. Vetter, K. Riley, K. Antypas, D. Bard, R. Coffey, E. Dart, S. Dosanjh, R. Gerber *et al.*, "Advanced scientific computing research exascale requirements review. an office of science review sponsored by advanced scientific computing research, september 27-29, 2016, rockville, maryland," Argonne National Lab.(ANL), Argonne, IL (United States). Argonne Leadership ..., Tech. Rep., 2017.

[13] I. Foster, M. Ainsworth, B. Allen, J. Bessac, F. Cappello, J. Y. Choi, E. Constantinescu, P. E. Davis, S. Di, W. Di *et al.*, "Computing just what you need: online data analysis and reduction at extreme scales," in *European Conference on Parallel Processing*. Springer, 2017, pp. 3–19.

[14] J. Y. Choi, C.-S. Chang, J. Dominski, S. Klasky, G. Merlo, E. Suchyta, M. Ainsworth, B. Allen, F. Cappello, M. Churchill *et al.*, "Coupling exascale multiphysics applications: Methods and lessons learned," in *2018 IEEE 14th International Conference on e-Science (e-Science)*. IEEE, 2018, pp. 442–452.

[15] J. Dominski, S.-H. Ku, C.-S. Chang, J. Choi, E. Suchyta, S. Parker, S. Klasky, and A. Bhattacharjee, "A tight-coupling scheme sharing minimum information across a spatial interface between gyrokinetic turbulence codes," *arXiv preprint arXiv:1806.05251*, 2018.

[16] P. W. Hamilton, P. Bankhead, Y. Wang, R. Hutchinson, D. Kieran, D. G. McArt, J. James, and M. Salto-Tellez, "Digital pathology and image analysis in tissue biomarker research," *Methods*, vol. 70, no. 1, pp. 59–73, 2014.

[17] A. Madabhushi, "Digital pathology image analysis: opportunities and challenges," *Imaging in Medicine*, vol. 1, no. 1, p. 07, 2009.

[18] L. A. Cooper, A. B. Carter, A. B. Farris, F. Wang, J. Kong, D. A. Gutman, P. Widener, T. C. Pan, S. R. Cholleti, A. Sharma *et al.*, "Digital pathology: Data-intensive frontier in medical imaging," *Proceedings of the IEEE*, vol. 100, no. 4, pp. 991–1003, 2012.

[19] J. Saltz, R. Gupta, L. Hou, T. Kurc, P. Singh, V. Nguyen, D. Samaras, K. R. Shroyer, T. Zhao, R. Batiste *et al.*, "Spatial organization and molecular correlation of tumor-infiltrating lymphocytes using deep learning on pathology images," *Cell reports*, vol. 23, no. 1, p. 181, 2018.

[20] P. Mobadersany, S. Yousefi, M. Amgad, D. A. Gutman, J. S. Barnholtz-Sloan, J. E. V. Vega, D. J. Brat, and L. A. Cooper, "Predicting cancer outcomes from histology and genomics using convolutional networks," *Proceedings of the National Academy of Sciences*, p. 201717139, 2018.

[21] M. Mustra, K. Delac, and M. Grgic, "Overview of the dicom standard," in *ELMAR, 2008. 50th International Symposium*, vol. 1. IEEE, 2008, pp. 39–44.

[22] D. A. Clunie, *DICOM structured reporting*. PixelMed Publishing, 2000.

[23] R. Singh, L. Chubb, L. Pantanowitz, and A. Parwani, "Standardization in digital pathology: Supplement 145 of the dicom standards," *Journal of pathology informatics*, vol. 2, 2011.

[24] S. Jodogne, E. Lenaerts, L. Marquet, C. Erpicum, R. Greimers, P. Gillet, R. Hustinx, and P. Delvenne, "Open implementation of dicom for whole-slide microscopic imaging," in *Proceedings,-12th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (Volume 6)*, 2017, pp. 81–87.

[25] "FY 2018 Stockpile Stewardship and Management Plan," https://www.energy.gov/nnsa/downloads/stockpile-stewardship-and-management-plan-ssmp.

[26] S. Panda, M. Rao, P. Thenkabail, and J. E. Fitzerald, *Remotely Sensed Data Characterization, Classification, and Accuracies*. CRC Press, 10 2015.

[27] M. Wolf, H. Abbasi, B. Collins, D. Spain, and K. Schwan, "Service augmentation for high end interactive data services," in *2005 IEEE International Conference on Cluster Computing*, Sep. 2005, pp. 1–11.

[28] P. Nillson, "Git is a purely functional data structure," https://blog.jayway.com/2013/03/03/git-is-a-purely-functional-data-structure/, 2013.

[29] B. Lynn, "Git magic," 2007, [Online; accessed 20-March-2018]. [Online]. Available: http://www-cs-students.stanford.edu/~blynn/gitmagic/

[30] M. Scott, S. J. Johnston, and S. J. Cox, "Metagit: Decentralised metadata management with git," *Information Systems*, vol. 65, pp. 78–92, 2017.

[31] L. Murta, V. Braganholo, F. Chirigati, D. Koop, and J. Freire, "noWorkflow: capturing and analyzing provenance of scripts," in *International Provenance and Annotation Workshop*. Springer, 2014, pp. 71–83.

[32] Y. L. Simmhan, B. Plale, and D. Gannon, "A survey of data provenance in e-science," *ACM Sigmod Record*, vol. 34, no. 3, pp. 31–36, 2005.

[33] L. Moreau, B. Ludäscher, I. Altintas, R. S. Barga, S. Bowers, S. Callahan, G. Chin Jr, B. Clifford, S. Cohen, S. Cohen-Boulakia *et al.*, "Special issue: The first provenance challenge," *Concurrency and computation: practice and experience*, vol. 20, no. 5, pp. 409–418, 2008.

[34] L. Moreau, B. Clifford, J. Freire, J. Futrelle, Y. Gil, P. Groth, N. Kwasnikowska, S. Miles, P. Missier, J. Myers *et al.*, "The open provenance model core specification (v1. 1)," *Future generation computer systems*, vol. 27, no. 6, pp. 743–756, 2011.

[35] Y. Gil, V. Ratnakar, and D. Garijo, "Ontosoft: Capturing scientific software metadata," in *Proceedings of the 8th International Conference on Knowledge Capture*. ACM, 2015, p. 32.

[36] W. Oliveira, L. M. Ambrósio, R. Braga, V. Ströele, J. M. David, and F. Campos, "A framework for provenance analysis and visualization," *Procedia Computer Science*, vol. 108, pp. 1592–1601, 2017.

[37] Y. Gil, D. Garijo, S. Mishra, and V. Ratnakar, "Ontosoft: A distributed semantic registry for scientific software," in *e-Science (e-Science), 2016 IEEE 12th International Conference on*. IEEE, 2016, pp. 331–336.

[38] H. Miao, A. Chavan, and A. Deshpande, "Provdb: Lifecycle management of collaborative analysis workflows." in *HILDA@ SIGMOD*, 2017, pp. 7:1–7:6.

[39] J. F. Pimentel, L. Murta, V. Braganholo, and J. Freire, "noworkflow: a tool for collecting, analyzing, and managing provenance from python scripts," *Proceedings of the VLDB Endowment*, vol. 10, no. 12, pp. 1841–1844, 2017.

[40] T. McPhillips, T. Song, T. Kolisnik, S. Aulenbach, K. Belhajjame, K. Bocinsky, Y. Cao, F. Chirigati, S. Dey, J. Freire *et al.*, "Yesworkflow: a user-oriented, language-independent tool for recovering workflow information from scripts," *arXiv preprint arXiv:1502.02403*, 2015.

[41] A. Prabhune, R. Stotzka, V. Sakharkar, J. Hesser, and M. Gertz, "Metastore: an adaptive metadata management framework for heterogeneous metadata models," *Distributed and Parallel Databases*, vol. 36, no. 1, pp. 153–194, 2018.

[42] A. Bhardwaj, S. Bhattacherjee, A. Chavan, A. Deshpande, A. J. Elmore, S. Madden, and A. G. Parameswaran, "Datahub: Collaborative data science & dataset version management at scale," *arXiv preprint arXiv:1409.0798*, 2014.

[43] A. Bhardwaj, A. Deshpande, A. J. Elmore, D. Karger, S. Madden, A. Parameswaran, H. Subramanyam, E. Wu, and R. Zhang, "Collabora-

tive data analytics with datahub," *Proceedings of the VLDB Endowment*, vol. 8, no. 12, pp. 1916–1919, 2015.

[44] M. Maddox, D. Goehring, A. J. Elmore, S. Madden, A. Parameswaran, and A. Deshpande, "Decibel: The relational dataset branching system," *Proceedings of the VLDB Endowment*, vol. 9, no. 9, pp. 624–635, 2016.

[45] A. Chavan, S. Huang, A. Deshpande, A. Elmore, S. Madden, and A. Parameswaran, "Towards a unified query language for provenance and versioning," *arXiv preprint arXiv:1506.04815*, 2015.

[46] S. Bhattacherjee and A. Deshpande, "Rstore: efficient multiversion document management in the cloud," in *Proceedings of the 2017 Symposium on Cloud Computing*. ACM, 2017, pp. 658–658.