Software Defined Cyberinfrastructure

Ian Foster[1,3], Ben Blaiszik[1], Kyle Chard[1], and Ryan Chard[2]

[1]Computation Institute, University of Chicago and Argonne National Laboratory
[2]Computing, Environment, and Life Sciences, Argonne National Laboratory
[3]Department of Computer Science, The University of Chicago

***Abstract*—Within and across thousands of science labs, researchers and students struggle to manage data produced in experiments, simulations, and analyses. Largely manual research data lifecycle management processes mean that much time is wasted, research results are often irreproducible, and data sharing and reuse remain rare. In response, we propose a new approach to data lifecycle management in which researchers are empowered to define the actions to be performed at individual storage systems when data are created or modified: actions such as analysis, transformation, copying, and publication. We term this approach software-defined cyberinfrastructure because users can implement powerful data management policies by deploying rules to local storage systems, much as software-defined networking allows users to configure networks by deploying rules to switches. We argue that this approach can enable a new class of responsive distributed storage infrastructure that will accelerate research innovation by allowing any researcher to associate data workflows with data sources, whether local or remote, for such purposes as data ingest, characterization, indexing, and sharing. We report on early experiments with this approach in the context of experimental science, in which a simple if-trigger-then-action (IFTA) notation is used to define rules.**

I. Introduction

Improving research data lifecycle management practices is essential to enhancing researcher productivity, facilitating reproducible research, and encouraging collaboration [1], [2]. Yet, despite a rich collection of tools for such purposes as assigning persistent identifiers to data [3]–[5], creating descriptive metadata [4], [6], aggregating data and code to form research objects [7], providing reliable and high-performance data access [8], enforcing access control [**?**], [9], performing routine analyses via workflow tools [10], [11], and ensuring persistent storage [12], [13], it remains difficult for individual researchers, laboratories, and even institutions to implement and generalize such practices [14]–[16].

One reason for this disconnect between theory and practice is the distributed and heterogeneous nature of many research environments. When data are created, analyzed, and stored on computers and storage systems that span units and institutions, enforcing good practices can be challenging. But, what if a researcher or project leader could specify once, via simple rules, the actions to be performed when data is created or modified, without needing to reconfigure the software and workflows on every computer that touches that data? They could easily specify, for example, that every dataset created in their lab be registered in a catalog, that new files be automatically distributed to collaborators, that important data be encrypted and backed up to offsite storage, that newly acquired data be analyzed for quality and features automatically extracted, or that files untouched for several months be moved to archival storage. Good practices would be applied automatically, data sharing and reproducible research would become routine, and discovery would be accelerated. In other words, we would **transform the humble storage system from what is often a static data graveyard to an active, *responsive storage* device in which a myriad of operations can be applied to data automatically**, under user control.

While others have applied rules to data management problems [17]–[21], our approach is differentiated by its simple *recipe* notation, usable by non-experts, and by its decoupling of recipes from data management and storage technologies, enabling recipes to be associated with *any* storage system (e.g., object stores, parallel and distributed file systems, commercial cloud storage) and to interact with any data management, analysis, or transformation service with a container or REST interface. We term the resulting system **software-defined cyberinfrastructure (SDCI)** because, as in software defined networks (SDN) [22], user policies are implemented via simple distributed mechanisms, albeit mechanisms deployed at the storage system rather than in the router as in SDN. The result is a cyber-environment in which data can easily be pushed to where it is needed and processed accordingly.

We believe that this SDCI approach has broad applicability across many disciplines and environments.

II. Goals and approach

The swift adoption of computational and data-driven research practices has resulted in increasingly intricate data lifecycles. We see the taming of this complexity as an important use case for SDCI.

A. The research data lifecycle

The research data lifecycle encompasses the instruments, operations, storage locations, and users that interact with data from when they are first acquired or created through to when they are finally archived or deleted.

Data may be created from a simulation, acquired from an instrument, or derived from other data. It is not unusual for a single research laboratory to operate tens or even hundreds of scientific instruments and for those instruments and associated computational projects to generate many thousands of datasets per year. The resulting data may then be stored in

a researcher's laptop, flash drive, cloud storage, departmental cluster, collaborator's PC, publication repository, or archival tape storage. Most data are stored in more than one location over time as priorities evolve, for example from fast online access to cheap archival storage. Data may undergo processing to validate quality, impute missing data, fix inconsistencies or errors, transform to different representations, extract information for indexing, compare with other data, subset, aggregate, model, or derive new results. Data will also be shared, replicated, and exchanged with a variety of other researchers, administrators, curators, publishers, and students.

The storage, management, and analysis practices used for different datasets and in different labs and communities vary widely, but are predominately ad hoc and manual, with negative consequences for researcher productivity, data sharing, data discovery, data reuse, experimental reproducibility, and ultimately scientific discovery. Inevitably, researchers spend a lot of time moving data among various systems, processing data with a number of different tools, and keeping track of data location and status.

B. Automation of data lifecycles

We believe that considerable parts of this lifecycle can be automated. Certainly, many activities listed are both frequent and common across a broad set of researchers. It is these activities that we focus on automating, in particular the automated transformation, extraction, sharing, analysis, replication, archival, and publication of data as a result of events such as data creation, arrival, and modification. These operations capture a large portion of the research data lifecycle.

Our overarching goal is to reduce barriers to establishing effective data lifecycle processes within the context of such facilities and projects. To this end, we intend to make it straightforward for individual researchers, project teams, facility administrators, and others to define and deploy end-to-end research data management *recipes* that will ensure that, for example, data produced by an instrument are subject to appropriate quality control, converted to a standard format, and registered in a catalog; that when new data appear in a catalog, notifications are dispatched to other (e.g., public) registries; and that data that are to be published are assigned appropriate identifiers and placed in appropriate archival storage.

The key to achieving this goal, we believe, is to deploy new SDCI capabilities on storage systems within research labs (to enable detection of events and invocation of recipes) and on computing resources wherever they are located (for execution of data lifecycle processes). In so doing, we can **accelerate discovery** by automating mundane data management processes, such as data placement, feature extraction, and transformation; **enhance reliability, security, and process transparency** by integrating high-quality, secure auditing and access control mechanisms into workflows; and **encourage data sharing and collaboration** by streamlining processes by which researchers can catalog, transfer, and replicate data.

For example, consider a situation in which researchers collect scanning electron microscope (SEM) data. They then want these data to be transformed to a particular format, indexed in a catalog for subsequent management using metadata relevant to SEM experiments, and replicated to an offsite storage system, after which researchers in the collaboration should be notified of the data creation. Collaborators, in turn, may then have their own processes for incorporating data in their work. While each of these steps may be individually straightforward, at present they are often managed by groups of researchers, postdocs, or students who manually advance the data through the lifecycle. Using the SDCI approach a researcher could define a recipe that states that on creation of a new image file, an action `extract_SEM` should be run to extract metadata relevant to SEM experiments. This action can in turn trigger other actions that register the experiment in an online catalog, transfer files with Globus, and notify people via email. Figure 1 shows other examples of automation.

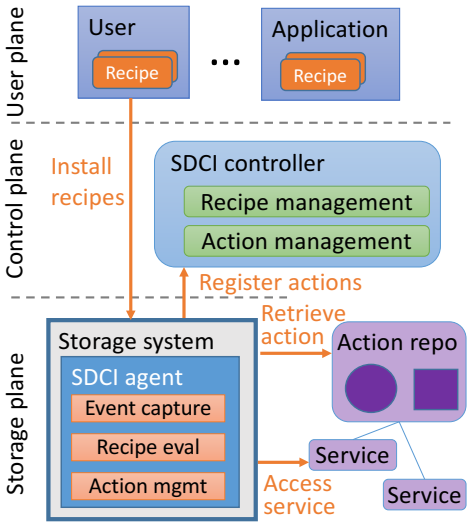

Fig. 2: SDCI architecture. Users install recipes on storage systems to implement specific data management policies. Events trigger actions (based on definitions in a repository), which may invoke services.

III. A Responsive Storage Fabric

As shown in Figure 2, we believe that we can realize the proposed SDCI approach by implementing a new *responsive storage* capability that will allow file system and other events to trigger data management/manipulation actions, and through these capabilities serve as the SDCI controller connecting storage systems, data portals (e.g., the Materials Data Facility [13]), and data manipulation tools and services (e.g., transformations supported by the Brown Dog service [23], Dockerized tools, command line tools). We will need an intuitive *recipe specification model and evaluation engine* that will allow users to define how such connections are triggered by defining recipes in terms of *events* and *actions*. We also need to produce a *data action toolkit* comprising a rich set of data manipulation services and tools, both domain-specific

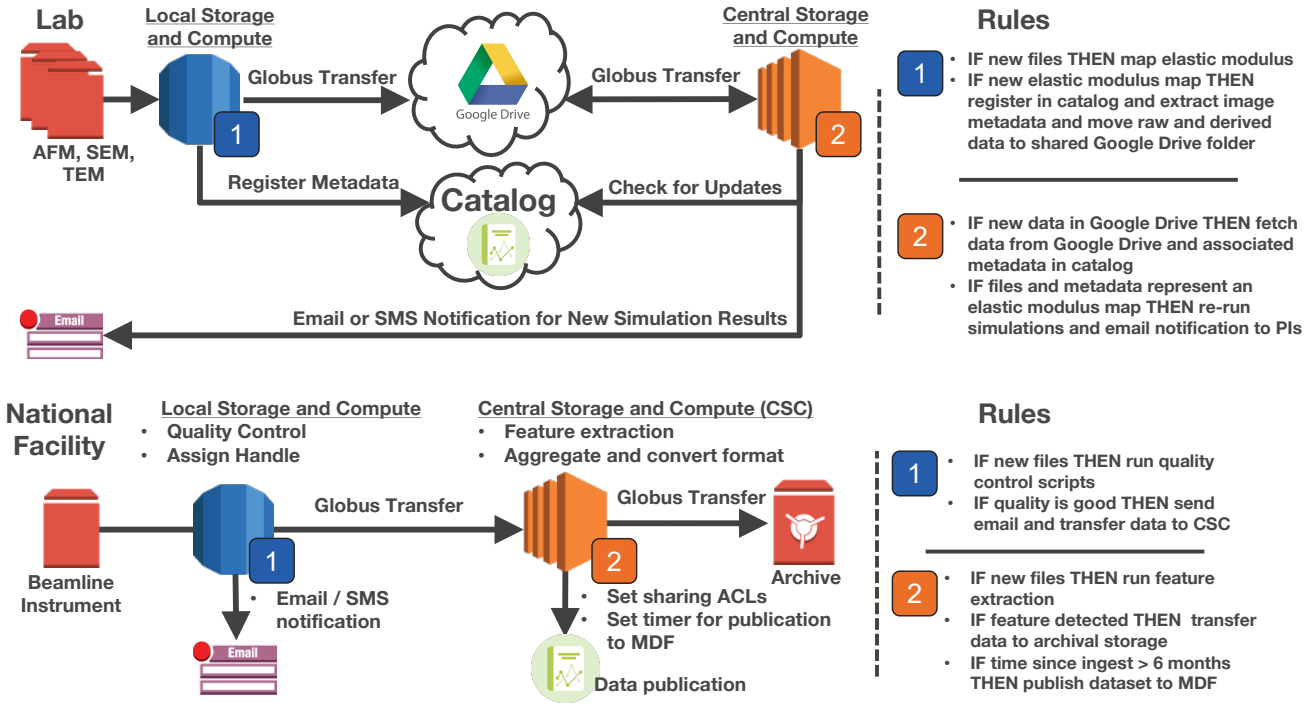

Fig. 1: Two example applications of the SDCI approach. Above: Data generated in a lab, for example by an atomic force microscope (AFM), is processed to compute elastic modulus; upon success, further processing and cataloging are performed, and data are transferred to Google Drive, which in turn spurs further actions at the central facility. Below: Data from a national facility is quality controlled, analyzed, archived, etc., according to the specified rules.

and domain-independent, to serve scientific data management needs.

As a basis for enabling such automated actions we require a low-level fabric that is able to detect and respond to storage system events. In particular, we require:

1) the ability to detect and be notified of file system events, such as creation, modification, or deletion of files or directories;
2) a model for representing and evaluating recipes based on arbitrary events and with arbitrary actions;
3) an action management module that can respond to events in ways defined by user-specified "handlers": for example, running a specified suite of extractions, or moving data to a secure location to meet data embargo requirements or to a facility with capacity sufficient for large data or big computation; and
4) an execution interface to execute actions locally in sandboxed containers. This model will ideally implement an SDN-like decoupling of control plane and storage plane.

A. Security model

One primary concern that permeates the responsive storage model is the need for fine-grained security. We must allow administrators to restrict (i.e., via inclusion/exclusion criteria) the paths that can be monitored and the actions that can be executed. Users, via recipes, must then be able to further constrain the events on which they want to act by defining paths and event types in their recipes. Such restrictions must be managed at a low level to ensure that events are filtered before any potential release of information.

We require a cohesive security fabric that extends to the action execution environment, so as to ensure that only permitted actions are executed on a storage system and that data is only released to external actions with strict user-oriented control. A delegated authentication and authorization model is needed to ensure that all components operate in a secure manner with explicit authorization [24].

B. Capturing events

Our responsive storage model requires a scalable, file-oriented event capture system to detect events as they occur (e.g., file creation) and to communicate them to the control pane (via an event bus) in a standard format, so that registered recipes can be matched and registered handlers invoked to perform actions. While this system is primarily envisaged to be used for file system events, we also want to allow for the integration of other event types, such as those generated by timers or external services.

Given the increasingly broad range of storage architectures used by researchers—encompassing traditional file systems, internal and external object storage, and high performance and archival storage—we design our approach to support a variety of different event interfaces, with the requirement that

```
{
  "recipe": [
    {
      "event-channel" : "FILESYSTEM",
      "event-type" : ["CREATION", "MODIFICATION"],
      "event-path" : ["/home/smith/"],
      "condition" : "*.tiff",
      "action-handler" : "MyHandler",
      "action-type" : ["TRANSLATE"],
      "action-options" ": {"format" : "png"}
    }
  ]
}
```

Fig. 3: An example recipe

a lightweight agent is deployed to serve as a gatekeeper to the specific storage system.

We aim to avoid file system modifications or the use of user-space file systems (e.g., FUSE [25]) due to inefficiencies and deployment difficulties. Instead, we propose to leverage existing storage system notification mechanisms. For example, Linux file systems can be monitored via the Linux inotify API [26] and Box supports the webhooks API [27]. Other systems have similar interfaces.

The agent deployed on storage systems will receive events directly from the storage system, filter based on registered events, and publish results to an event bus. As some events may occur with high frequency, we will investigate the use of tiering and batching, and of prioritizing updates that match defined recipes. For example, changes to file size may be less urgent to convey than file deletion or creation.

C. Specifying and evaluating recipes

A growing number of consumer systems (e.g., for home automation) support simple if-trigger-then-action (IFTA) *recipes*, as popularized by the "if this then that" (IFTTT) service [28], which supports hundreds of event sources ("channels") and action types. While expressiveness is limited (one event triggers one action), studies show that people find this model intuitive [29], [30]. We believe that it has promise for specifying the actions to be performed in response to an event.

In the IFTA model, a recipe comprises: 1) a registered *event* from a specified channel (e.g., file system); 2) a *condition* from which a matching recipe can be selected (e.g., regular expression); and 3) a registered *handler* that abstracts the execution of an action (e.g., transformation). We adopt a flexible and extensible JSON-based representation for recipes. Figure 3 shows an example recipe that monitors a file system for creation and modification of *tiff* files and invokes a transformation to *png* format. Figure 1 shows additional examples.

The notation shown here has the advantage of simplicity. As is done in IFTTT, we can provide a graphical interface to enable users to select supported events and handlers, author recipes by selecting or defining conditions, and discover and share recipes. A research topic will be to study expressiveness–simplicity tradeoffs, and to explore and evaluate extensions, such as more expressive condition logic, arithmetic and string manipulation functions, core functions (e.g., path manipulation), and recipe chaining.

TABLE I: Examples of events and, for each, a system that can be used to generate notifications

Event	Notifier
File creation	File system
File update	File system
File deletion	File system
Access control list (ACL) change	Globus
Transfer event (e.g., complete, error)	Globus
Periodic event (e.g., every day)	Timer
Time since (e.g., since file touched)	Timer
User annotation read, write, update	Catalog
Metadata curation	Catalog

TABLE II: Examples of actions and, for each, a component that might be invoked to perform that action

Action	System
Extract metadata	Transformation service
Register metadata	Catalog
Convert format	Transformation service
Execute transformation	Execution
Transfer, share, replicate	Globus
Share	Globus
Publish	Globus & others
Replicate/mirror	Globus
Archive or extract	Zip/Tar
Extract	Zip/Tar
Notify	Email
Job submission	Job queue
Start timer	Timer

In order to execute recipes, we will need to provide users with a set of event channels and handlers. Presumably administrators will be able to restrict the event channels, event types, handlers, and actions allowed on their endpoints. Tables I and II list some events and actions that we might want to support.

D. Action management

Once a recipe matches an event, we use three components to execute the associated action: 1) a registered handler that describes how an action is invoked, 2) an execution environment for executing actions locally or invoking remote services, and 3) a repository for storing and sharing actions.

A handler may invoke a service (e.g., to transfer data) or execute an operation on the storage system (e.g., metadata extraction). Each handler may require different parameters in order to execute a given action. For instance, a handler that dispatches a file to a transformation service requires the file to be translated and the format to be translated to. To ensure that this information is provided, we define a mechanism for registering handler definitions using a lightweight templated JSON representation (Figure 4). This representation enables multiple actions to be defined for a given handler, arbitrary

```
{
 "handler": {
   "name": "MyHandler"
   "actions": [
     {
       "name": "TRANSLATE",
       "url": "https://bd-api.org/convert/$format"
       "arguments": [
          {"name": "file_name", "type": "file",
           "action" : "upload"},
          {"name":  "type", "type": "string",
           "replace" : "$format"}
       ]
     }  ...
   ]
 }
}
```

Fig. 4: Example handler registration

arguments to be passed to an action, and arguments to be templated into invocation commands (e.g., in a REST URL).

For service-based actions, the service will be invoked via the handler's REST URL. For tasks to be invoked locally (e.g., a script or other program) we will provide a lightweight container-based execution environment. Administrators will be able to manage an execution queue to ensure that resources are not overwhelmed and that containers operate within appropriate bounds (e.g., execution limits). This model does not require a complete job submission system, but rather just a local API that can be used to execute containers.

We may want to base our container execution environment on Docker [31], due to its rich ecosystem of supporting software for composing, managing, and executing containers. Docker allows users to wrap arbitrary applications in an environment for isolated execution. Containers have four advantages for our purposes: 1) applications execute in the same way, irrespective of the host ecosystem, facilitating reproducible research [32], 2) they can be created and destroyed quickly, 3) they are isolated from the host system and also from one another [33], and 4) they can be easily moved to new hosts.

E. Managing recipe execution

Users and administrators must be able to determine when actions succeed or fail. We propose to use a local manager to monitor each action execution to ensure that it executes to completion or failure. If actions become unresponsive, it will terminate the running action and attempt to re-execute it. Several such terminations will result in an error being recorded and users notified. We will record all actions executed and the conditions that triggered each execution, and allow authorized users to access this history.

Thus, users and administrators will be able to interrogate the actions performed on their storage system(s), review events that occurred, and determine the results of action execution. They will be able to view failed actions, cancel an action, or replay an action at a later time if the fault is subsequently resolved.

IV. Experiments

To validate the SDCI model we have created a prototype system called Ripple [34] and applied it to two use cases.

A. The Ripple system

Ripple is implemented as a Python-based agent deployed to local file systems and a web service that manages the execution of actions in response to events.

The local agent relies on the Python-based Watchdog module which defines a standard interface for capturing (file system) events. It can be extended via a standard interface such that external events (e.g., those created by a service) may also be captured. The agent also includes a SQLite database that stores all active recipes and provides an efficient means of evaluating events against active recipes. When events are captured, they are compared with active recipes, any that match are then passed to the cloud service for evaluation. We use an Amazon Simple Notification Service (SNS) topic (a reliable queue) to buffer events.

The cloud service reads from the SNS topic and performs actions accordingly. To do so, it uses an Amazon Lambda function to evaluate the event and the matched recipe. Depending on the defined action, one of several things may happen. Arbitrary Lambda functions can be used to perform an action; external services can be invoked by a Lambda function (e.g., to transfer data using Globus or send an email using Amazon's Simple Messaging Service); or a container can be executed on the local storage system. In the latter case, the Ripple agent includes an interface for managing the execution of containers. This interface, essentially a light wrapper over the Docker CLI, enables registered containers to be executed with arbitrary parameters passed through from the Lambda function.

We have also developed a lightweight web interface through which users can view active recipes, activate/deactivate recipes, or define a new recipe on a particular storage system. It also provides an audit log such that users can view which recipes have been fired and the outcomes of their executions.

B. Application of Ripple

To explore the capabilities of Ripple we have deployed two testbeds in which we emulate the processing of data from two different experimental sources: the Large Synoptic Survey Telescope (LSST) [35] and the Advanced Light Source (ALS), a synchrotron light source at Lawrence Berkeley National Laboratory.

In the first example, we deployed an Amazon public cloud-based testbed to emulate the observatory (data source) and two custodial stores in Chile and the US. This Amazon-based infrastructure is combined with an archival storage system at Argonne National Laboratory. We deployed Globus endpoints on each system to enable remote transfer of data between storage systems. We then implemented a number of rules and deployed them to each node in the testbed.

Collectively, these rules support the following lifecycle: (1) The telescope generates a FITS format image. RIPPLE filters

file system events within the source directory to discover FITS files. (2) Data are uploaded directly to the Chilean custodial store using HTTP. (3) Docker containers are executed to extract metadata and catalog the data in a global file registry. (4) A unique identifier is generated for the data by using the Minid service [36]. (5) Data are synchronized to the US custodial store, where similar metadata processing occurs. (6) Data are compressed as a gzip file. (7) Finally, data are transferred to the archival data store at Argonne.

In the second example, we deployed a Ripple agent on a machine at the ALS and also on a login node associated with a supercomputer at the National Energy Research Scientific Computing Center (NERSC). We then simulated data being generated at an ALS beamline and implemented recipes to manage the data lifecycle from creation through dissemination, as follows. (1) A new HDF5 file created by a beamline experiment is detected on the ALS machine. (2) The file is transferred to NERSC using Globus. (3) A metadata file and a batch submission file are created based on the input file. (4) The batch submission file triggers a job to be submitted to the Edison supercomputer. (5) When the results are detected from the compute job, the results are transferred back to the ALS machine. (6) A Globus shared endpoint is created and the raw and derived data are shared with a predefined group of collaborators. (7) Finally, an email is sent to all collaborators notifying them of completion.

These experiments provide some initial evidence of the promise of the approach, although wider deployment and much more experimentation are required to reach firm conclusions.

V. Related Work

Previous rule-based approaches to data management [37] are primarily designed for expert administration of large data stores. Our approach is differentiated by its simple recipe notation and decoupling of rules from data management and storage technologies. Below we discuss several rules engines and discuss how they relate to our work.

In database systems, production rules have long been used to specify conditional actions [38]. Also relevant are RuleML [17], the Semantic Web Rule Language (SWRL) [39], and Drools [18]. These systems vary in their expressiveness, generality, and complexity. All are intended for expert users.

The integrated Rule-Oriented Data System (iRODS) [21] uses a powerful rules engine to manage the data lifecycle of the files and stores that it governs. iRODS is a closed system: data are imported into an iRODS data grid that is managed entirely by iRODS and administrators use rules to configure the management policies to be followed within that data grid. In contrast, we advocate for an open system: any authorized user may associate rules with any storage system.

IOBox [40] is designed to extract, transform, and load data into a catalog. It is able to crawl and monitor a file system, detect file changes (e.g., creation, modification, deletion), and apply pattern matching rules against file names to determine what actions (ETL) should be taken. The SDCI approach extends this model by allowing scalable and distributed event detection, and supporting an arbitrary range of actions.

Systems such as the Robinhood Policy Engine [41] are designed to manage large HPC file systems like Lustre. Robinhood maintains a database of file metadata and allows bulk actions, such as migrating or purging stale data, to be scheduled for execution against collections of files. It also provides routines to manage and monitor file systems efficiently, such as those used to find files, determine usage, and produce reports. It is not the goal of SDCI to provide such utilities. Instead, we aim to empower users to implement simple, yet effective data management strategies.

Specialized software is also used to manage common flows within particular domains. For example, SPADE [42] is designed to support automated transfer and transformation of data as it is created by an experiment. Users configure a SPADE *dropbox*. If a file is written to the dropbox, SPADE creates (or detects) an accompanying *semaphore file* to signal that the file is complete and that a transfer should begin. SPADE can also enable data archival or execution of analysis scripts in response to data arrival. The SPOT framework [43] is a workflow management solution developed specifically for the Advanced Light Source at Lawrence Berkeley National Laboratory. SPOT leverages SPADE to automate the analysis, transfer, storage, and sharing, of ALS users' data using HPC resources. In contrast to SPOT's pre-defined flows which handle large data volumes and numbers of data sets, the SDCI model aims to empower non-technical users to define custom recipes that can be combined into adaptive flows.

VI. Summary

We have proposed software-defined cyberinfrastructure (SDCI) as a new approach to managing the research data lifecycle across a diverse and distributed storage landscape, via the use of intuitive rule-based abstractions. We are currently engaged in early experiments with this approach, with so far promising results.

Acknowledgments

This work was supported in part by the U.S. Department of Energy under contract DE-AC02-06CH11357. We are grateful to Steve Tuecke, Kenton McHenry, Klara Nahrstedt, and others for helpful discussions on these topics.

References

[1] D. Atkins, T. Hey, and M. Hedstrom, "National Science Foundation Advisory Committee for Cyberinfrastructure Task Force on Data and Visualization Final Report," National Science Foundation, Tech. Rep., 2011, https://www.nsf.gov/cise/aci/taskforces/TaskForceReport_Data.pdf.

[2] V. Stodden, F. Leisch, and R. D. Peng, *Implementing reproducible research*. CRC Press, 2014.

[3] S. Sun, L. Lannom, and B. Boesch, "Handle system overview," IETF RFC 3650, Tech. Rep., 2003.

[4] "DataCite," https://www.datacite.org/. Visited August 12, 2016.

[5] "The Document Object Identifier System," https://www.doi.org/. Visited August 12, 2016.

[6] S. Weibel, J. Kunze, C. Lagoze, and M. Wolf, "Dublin core metadata for resource discovery," Internet Engineering Task Force, RFC 2413, 1998.

[7] S. Bechhofer, I. Buchan, D. D. Roure, P. Missier, J. Ainsworth, J. Bhagat, P. Couch, D. Cruickshank, M. Delderfield, I. Dunlop, M. Gamble, D. Michaelides, S. Owen, D. Newman, S. Sufi, and C. Goble, "Why linked data is not enough for scientists," *Future Generation Computer Systems*, vol. 29, no. 2, pp. 599 – 611, 2013, special section: Recent advances in e-Science. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X11001439

[8] I. Foster, "Globus Online: Accelerating and democratizing science through cloud-based services," *IEEE Internet Computing*, vol. 15, no. 3, p. 70, 2011.

[9] P. Samarati and S. C. de Vimercati, "Access control: Policies, models, and mechanisms," in *Foundations of Security Analysis and Design: Tutorial Lectures*, R. Focardi and R. Gorrieri, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 137–196. [Online]. Available: http://dx.doi.org/10.1007/3-540-45608-2_3

[10] J. Goecks, A. Nekrutenko, and J. Taylor, "Galaxy: A comprehensive approach for supporting accessible, reproducible, and transparent computation research in the life sciences," *Genome Biology*, 2010.

[11] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. F. da Silva, M. Livny, and K. Wenger, "Pegasus, a workflow management system for science automation," *Future Generation Computer Systems*, vol. 46, pp. 17–35, 2015.

[12] G. King, "An introduction to the Dataverse Network as an infrastructure for data sharing," *Sociological Methods & Research*, vol. 36, no. 2, pp. 173–199, 2007.

[13] B. Blaiszik, K. Chard, J. Pruyne, R. Ananthakrishnan, S. Tuecke, and I. Foster, "The Materials Data Facility: Data services to advance materials science research," *Journal of Materials*, vol. 68, no. 8, pp. 2045–2052, 2016.

[14] J. P. Birnholtz and M. J. Bietz, "Data at work: Supporting sharing in science and engineering," in *International ACM SIGGROUP Conference on Supporting Group Work*. ACM, 2003, pp. 339–348.

[15] C. J. Savage and A. J. Vickers, "Empirical study of data sharing by authors publishing in PLoS journals," *PloS one*, vol. 4, no. 9, p. e7078, 2009.

[16] A. F. Magee, M. R. May, and B. R. Moore, "The dawn of open access to phylogenetic data," *PLoS One*, vol. 9, no. 10, p. e110268, 2014.

[17] A. Paschke, "RBSLA a declarative rule-based service level agreement language based on RuleML," in *International Conference on Computational Intelligence for Modelling, Control and Automation*, vol. 2. IEEE, 2005, pp. 308–314.

[18] P. Browne, *JBoss Drools business rules*. Packt Publishing Ltd, 2009.

[19] A. Rajasekar, R. Moore, and F. Vernon, "iRODS: A distributed data management cyberinfrastructure for observatories," in *AGU Fall Meeting Abstracts*, vol. 1, 2007, p. 1214.

[20] R. W. Moore and A. Rajasekar, "Rule-based distributed data management," in *IEEE/ACM International Conference on Grid Computing*, 2007.

[21] A. Rajasekar, R. Moore, C.-y. Hou, C. A. Lee, R. Marciano, A. de Torcy, M. Wan, W. Schroeder, S.-Y. Chen, L. Gilbert, P. Tooby, and B. Zhu, "iRODS Primer: Integrated rule-oriented data system," *Synthesis Lectures on Information Concepts, Retrieval, and Services*, vol. 2, no. 1, pp. 1–143, 2010.

[22] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[23] S. Padhy, J. Alameda, R. Liu, E. Black, L. Diesendruck, M. Dietze, G. Jansen, P. Kumar, R. Kooper, J. Lee, R. Marciano, L. Marini, D. Mattson, B. Minsker, D. Navarro, M. Slavenas, W. Sullivan, J. Votava, I. Zharnitsky, and K. McHenry, "Brown Dog: An elastic data cyberinfrastrure for autocuration and digital preservation," in *SEA Software Engineering Assembly*, 2016.

[24] S. Tuecke, R. Anathankrishnan, K. Chard, M. Lidman, B. McCollam, S. Rosen, and I. Foster, "Globus Auth: A research identity and access management platform," in *16th International Conference on e-Science*, 2016.

[25] "FUSE: Filesystem in userspace," http://fuse.sourceforge.net. Accessed August 1, 2016.

[26] "Linux programmers manual: inotify API," http://man7.org/linux/man-pages/man7/inotify.7.html. Accessed August 1, 2016.

[27] "Box Webhooks," https://docs.box.com/docs/webhooks. Visited August 10, 2016.

[28] "If This Then That," http://www.ifttt.com. Visited August 1, 2016.

[29] B. Ur, E. McManus, M. Pak Yong Ho, and M. L. Littman, "Practical trigger-action programming in the smart home," in *SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2014, pp. 803–812.

[30] A. K. Dey, T. Sohn, S. Streng, and J. Kodama, "iCAP: Interactive prototyping of context-aware applications," in *International Conference on Pervasive Computing*. Springer, 2006, pp. 254–271.

[31] D. Merkel, "Docker: Lightweight Linux containers for consistent development and deployment," *Linux Journal*, no. 239, p. 2, 2014.

[32] C. Boettiger, "An introduction to Docker for reproducible research," *ACM SIGOPS Operating Systems Review*, vol. 49, no. 1, pp. 71–79, 2015.

[33] T. Bui, "Analysis of docker security," *arXiv preprint arXiv:1501.02967*, 2015.

[34] R. Chard, K. Chard, J. Alt, D. Y. Parkinson, S. Tuecke, and I. Foster, "Ripple: Home automation for research data management," in *1st International Workshop on Serverless Computing*, 2017.

[35] Z. Ivezic, J. Tyson, B. Abel, E. Acosta, R. Allsman, Y. AlSayyad, S. Anderson, J. Andrew, R. Angel, G. Angeli *et al.*, "LSST: From science drivers to reference design and anticipated data products," *arXiv preprint arXiv:0805.2366*, 2008.

[36] K. Chard, M. D'Arcy, B. Heavner, I. Foster, C. Kesselman, R. Madduri, A. Rodriguez, S. Soiland-Reyes, C. Goble, K. Clark, E. W. Deutsch, I. Dinov, N. Price, and A. Toga, "I'll take that to go: Big data bags and minimal identifiers for exchange of large, complex datasets," in *IEEE International Conference on Big Data*, Washington, DC, USA, 2016.

[37] A. Rajasekar, M. Wan, R. Moore, and W. Schroeder, "A prototype rule-based distributed data management system," in *HPDC Workshop on Next Generation Distributed Data Management*, vol. 102, 2003.

[38] E. N. Hanson and J. Widom, "An overview of production rules in database systems," *The Knowledge Engineering Review*, vol. 8, no. 02, pp. 121–143, 1993.

[39] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean, "SWRL: A semantic web rule language combining OWL and RuleML," *W3C Member submission*, vol. 21, p. 79, 2004.

[40] R. Schuler, C. Kesselman, and K. Czajkowski, "Data centric discovery with a data-oriented architecture," in *1st Workshop on The Science of Cyberinfrastructure: Research, Experience, Applications and Models*, ser. SCREAM '15. New York, NY, USA: ACM, 2015, pp. 37–44.

[41] T. Leibovici, "Taking back control of HPC file systems with Robinhood Policy Engine," *arXiv preprint arXiv:1505.01448*, 2015.

[42] "SPADE," http://nest.lbl.gov/projects/spade/html/. Accessed March 1, 2017.

[43] J. Deslippe, A. Essiari, S. J. Patton, T. Samak, C. E. Tull, A. Hexemer, D. Kumar, D. Parkinson, and P. Stewart, "Workflow management for real-time analysis of lightsource experiments," in *9th Workshop on Workflows in Support of Large-Scale Science*. IEEE Press, 2014, pp. 31–40.