Science Capsule: Towards Sharing and Reproducibility of Scientific Workflows

Devarshi Ghoshal*, Ludovico Bianchi*, Abdelilah Essiari*, Drew Paine*,
Sarah S. Poon*, Michael Beach†§, Alpha T. N'Diaye*, Patrick Huck*, Lavanya Ramakrishnan*
*Lawrence Berkeley National Laboratory, Berkeley, CA
†University of Washington, Seattle, WA
Email: [dghoshal, lbianchi, aessiari, pained, sspoon, atndiaye, phuck, lramakrishnan]@lbl.gov [mwb8]@uw.edu

***Abstract*—Workflows are increasingly processing large volumes of data from scientific instruments, experiments and sensors. These workflows often consist of complex data processing and analysis steps that might include a diverse ecosystem of tools and also often involve human-in-the-loop steps. Sharing and reproducing these workflows with collaborators and the larger community is critical but hard to do without the entire context of the workflow including user notes and execution environment. In this paper, we describe *Science Capsule*, which is a framework to capture, share, and reproduce scientific workflows. Science Capsule captures, manages and represents both computational and human elements of a workflow. It automatically captures and processes events associated with the execution and data life cycle of workflows, and lets users add other types and forms of scientific artifacts. Science Capsule also allows users to create 'workflow snapshots' that keep track of the different versions of a workflow and their lineage, allowing scientists to incrementally share and extend workflows between users. Our results show that Science Capsule is capable of processing and organizing events in near real-time for high-throughput experimental and data analysis workflows without incurring any significant performance overheads.**

I. Introduction

The growth of data from instruments, sensors, and experiments has resulted in complex processing and data analysis. Scientific workflows help [1] manage the process from data collection to analysis [2]–[4]. Users use a myriad of scripts, tools, and lab notebooks to capture and manage the entire scientific process. For example, a user at a light source often performs coarse-grained data analyses in real-time to decide on additional experiments. These workflows often need to run on HPC resources at scale. Subsequently after the experiment, the workflow needs to be rerun for finer-grained analyses and then shared with collaborators and possibly, even with the larger community. Today, users lack tools for capturing, sharing, and reproducing workflows in such contexts.

Workflow tools with integrated provenance collection mechanisms [5], [6] are not sufficient as the workflows are increasingly managed across heterogeneous software environments and ecosystems. Existing tools that enable scientific reproducibility [7]–[10] often capture system and design-level provenance separately, and do not include the human elements of a workflow. Provenance information is often incomplete and structured provenance tools often don't let us capture the unstructured elements of the workflow. For e.g., the contextual information (scripts, parameters, decisions, etc.) are not sufficiently captured for users to be able to revisit, re-run, and share their workflows, data, or their provenance. This is challenging for workflows using experimental and observational data like that from the light sources, where researchers reuse workflows for real-time data analyses to refine the experiment and for post-experiment data analyses. Previous work in recording human activities in provenance is also limited to simple annotations [11], [12] that is not sufficient to capture the scientific complexities and thought process behind an experiment design or a workflow. Thus, there is a need to capture the relevant context about a workflow and its artifacts along with run-time information; that can seamlessly integrate with existing scientific software ecosystems to enable sharing, reuse and reproduction of workflows and their data.

In this paper, we describe the design and implementation of **Science Capsule**, which captures the processing and data life cycle of a workflow across machines, organizations, people, and science domains. Instead of ensuring complete reproducibility, Science Capsule creates a 'journal' of workflow activities that users can inspect, analyze and enhance, while sharing and reproducing workflows. Science Capsule uses system monitors for automated capture of provenance which can be enhanced with user artifacts (e.g., design notes, experiment protocols, etc.). Additionally, Science Capsule provides a way to compare and inspect workflows over time. To this effect, Science Capsule supports the idea of 'workflow snapshots', akin to a time capsule that captures and preserves all relevant information of a workflow at a particular time. Science Capsule's automated provenance collection augmented with methods to capture user artifacts, workflow snapshots, and intuitive interfaces provides the necessary foundation that users need to share and reuse workflows.

Science Capsule includes an event processing engine that processes and reduces system-level events into workflow activities, and a snapshot manager that tracks and manages workflows over time. The event processing engine uses rule-based transformations for cleaning, filtering and aggregating event data. The snapshot manager creates 'workflow snapshots', which uses a Git-based version control system to keep track of workflow artifacts, and also provides support for tracking large amounts of workflow data. Science Capsule also

§Work performed during internship at Berkeley Lab.

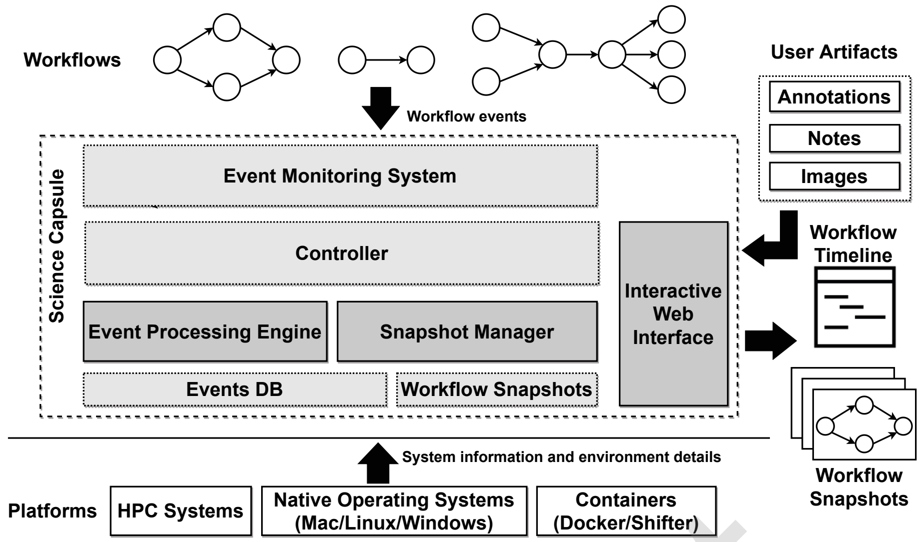


Fig. 1: Science Capsule contains three main components – a) event processing engine that transforms and reduces large amounts of raw event data, b) snapshot manager that manages and tracks workflows over time, and c) an interactive web interface that allows users to inspect, analyze and enhance workflow execution and provenance information. The controller handles incoming event data from the monitoring system, and responds to user requests.

provides an interactive web interface to view timeline and add artifacts to the workflow timeline. Science Capsule is capable of collecting and processing events at a high throughput and in real-time to address the needs of high-performance scientific workflows. We evaluate Science Capsule on two science workflows, and a synthetic benchmark across three systems to understand its broad applicability and performance overheads.

The rest of the paper is organized as follows. We discuss related work in Section II. We describe the design and implementation of Science Capsule in Section III, and provide the evaluation in Section IV. Finally, we present our conclusions in Section V.

II. Related Work

Over the past decade, several frameworks and programming interfaces have been developed to create reproducible packages of scientific workflows by capturing the metadata and provenance information [7]–[10]. Workflow management systems have also been instrumented to collect necessary metadata and provenance information [5], [6]. However, these tools require adapting existing user workflows and work practices to a specific software ecosystem, which is a challenge for scientific ecosystems. Also, they do not extensively capture the human elements of a workflow (e.g., lab notebooks), and are primarily limited to simple user annotations [11], [12]. Science Capsule collects workflow runtime and provenance information in a non-intrusive manner through system-level monitoring, while allowing users to enhance the information by managing user-defined artifacts as part of the workflow.

Metadata catalogs like iRODS [13] and data context service like Ground [14] collect and manage metadata about workflow data and processes. Bluesky data broker [15] provides an integrated interface to access data and metadata from light source experiments. Science Capsule can be easily integrated with such metadata management services to enrich the provenance derived from system-level monitoring.

Enabling reproducible scientific work is extremely challenging because it depends on a researcher's ability to capture, curate, and share the deluge of artifacts in usable ways [16]–[20]. Several methods have been proposed to extract and represent meaningful information from provenance traces of scientific workflows. Methods using data mining [21], [22], ontologies [23], [24] and W3C PROV [25] have been used to extract useful information from large workflow and provenance traces. Other tools and algorithms have been developed to compare and extract information from workflow executions and provenance traces over time [26]–[28]. Science Capsule not only represents a workflow's execution information, provenance and artifacts, but also their versions over time, allowing users to study, share, execute and reproduce different versions of the workflow through a single interface.

III. Design and Implementation

Figure 1 shows the high-level architecture of Science Capsule. There are three main components in Science Capsule– a) **event processing engine** that derives information about the workflow activities by aggregating and reducing system events, b) **snapshot manager** that allows users to navigate workflows across the time domain, and c) **an interactive web interface** that provides an interface to collect user feedback for enhancing workflow metadata and provenance. When a user initiates a Science Capsule, it starts monitoring and capturing the events from the execution and data life cycle of workflows. The processing engine filters and aggregates these events to generate provenance, context, and processing information about the workflows. It essentially transforms raw, unprocessed system events into workflow activity and provenance information;

creating a ‘workflow timeline’, which represents a journal of workflow activities that users can inspect, analyze and enhance through the interactive web interface. If users need to annotate specific workflow activities or add information outside of the computing life cycle of a workflow (e.g., lab notes, image scans etc.), they can do so through the web interface – Science Capsule adds and associates the information to the workflow timeline. In addition to recording and storing fine-grained events of workflow activities, Science Capsule also manages the different states of a workflow in its entirety, which allows users to rollback or skip forth to different workflow versions over time. In order to allow for time-based workflow navigation, the snapshot manager in Science Capsule saves, tracks, and manages the different states of a workflow and associated artifacts as ‘workflow snapshots’. It uses Git as the underlying version control system along with different storage back-ends (ObjectDB and Git-LFS) to efficiently track and manage workflow artifacts over time.

In Science Capsule, events from workflow executions are captured through an active monitoring system that can automatically capture both process (e.g., commands) and data events (e.g., files created, data written) – the amount and type of information collected depends on the underlying platform and the method of capture. Currently, Science Capsule provides two ways to capture and manage the events and provenance information from workflows – a) within a container (container mode) that allows Science Capsule to capture the entire workflow environment and life cycle, ensuring portability, reproducibility, and sharing of the containerized workflow, and b) on the native OS of a scientist’s laptop or compute cluster (native OS mode), where Science Capsule captures the execution time provenance on user-specified artifacts, which can be used for understanding, recreating, and sharing the workflow. The container mode allows Science Capsule to capture as much information as possible by monitoring the closed environment where the workflows run. The native mode allows Science Capsule to collect only selective information about the workflows as is required by the users.

A. *Events Management*

Science Capsule monitors, captures, processes and stores events from various sources during workflow execution. In order to monitor the workflow environment and collect events, users either enable the monitoring of workflow-specific directories explicitly (native OS mode), or use a container that has Science Capsule pre-installed (container mode).

Event collection. Science Capsule captures various system-level events using existing tools that enable cross-platform event monitoring and capturing of file system and process invocation events. Currently, system-level events are captured in Science Capsule using three tools – inotify [29], watchdog [30], and strace. Inotify and watchdog are used to capture file system events. Strace is a Linux utility to capture process related events in addition to file system events. By default, Science Capsule selects the monitoring tool based on the underlying platform. For example, inotify is enabled by default, when Science Capsule is monitoring workflows on Linux-based systems, whereas watchdog is enabled for Windows and MacOS users. In addition to the platform where the workflows run, users can select the monitoring tool based on the granularity of events they need to capture. For example, inotify only monitors file system events, whereas strace monitors both file system and process events.

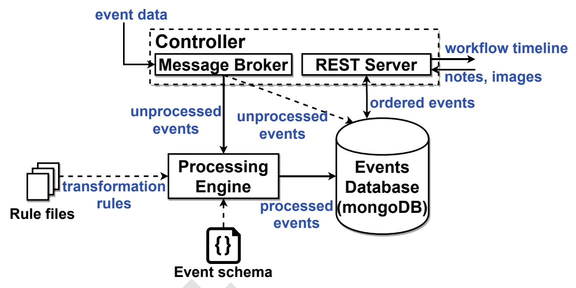


Fig. 2: The processing engine in Science Capsule receives unprocessed events from the controller through a message broker and stores processed events in a mongoDB collection. The REST server provides an interface for retrieving and adding workflow-specific events.

Event processing. Figure 2 shows the event processing in Science Capsule. Events from the different sources might be too fine-grained for users to make any meaningful interpretation about the workflow’s execution or data life cycle. For example, the file system events captured using inotify contain file system operations at the block-level, where a single file read operation might result in multiple file system events. The processing engine in Science Capsule filters, transforms and aggregates these low-level system events into a timeline of workflow activities describing the provenance, contextual data and processing information about the workflows. The system events are transformed and reduced using the time and data dependency information from the event data. The event times imply the order in which the different activities of a workflow happened during execution. The reduction is based on the number of files accessed by the workflow over time, and not the individual file operations (file open, read/write, close) as captured at the system-level. In case a file is opened, read multiple times and closed, Science Capsule creates a single workflow activity aggregating all the I/O statistics (number of bytes read, read start time, read end time etc.).

Adding external event sources. In addition to capturing workflow provenance and execution information from monitoring system events, Science Capsule can also capture information from other external event sources such as workflow scripts or workflow management systems, to enrich the metadata and provenance collected using the system-level monitoring tools. For example, a beamline scientist at the light sources might want to know the specific configurations used in their workflows. While the default monitoring system in Science Capsule captures the file usage and program invocations of a workflow, events from any analysis tool might describe additional steps at the application granularity.

In order to extract and store additional workflow execution

Listing 1 Example rule file for processing events.

```
{
 "timestamp": {
   "column": "time",
   "format": "%Y-%m-%dT%H:%M:%S.%f%z"
 },
 "cleaning" : {
   "must_have_columns" : [ "step",
                           "input",
                           "status" ],
   "rename_columns" : { "step" : "name",
                        "input": "vars"}
 },
 "mappings" : {
   "artifact" : "name",
   "action" : "status",
   "metadata" : [ "vars"]
 }
}
```

and provenance information into Science Capsule, the events from external sources need to be processed similar to system-level events. The processing involves transforming and selecting values from the event data that can be used by Science Capsule to augment additional information into the workflow timeline. However, events from the external sources can be of any format or type. To simplify processing events from different external sources, Science Capsule requires users to provide configuration files specifying simple rules to extract workflow information from event data. This also allows Science Capsule to process events for any workflow without the need to rewrite or adapt them to a specific workflow engine or tool. Listing 1 shows an example configuration file. Currently, these configuration files may contain simple rules for cleaning and processing events such as formatting time values, filtering out non-essential fields and mapping input data to an event in Science Capsule. However, they may be extended to support more complex data cleaning and processing for any arbitrary event source.

B. Workflow Tracking and Snapshots

Figure 3 highlights workflow tracking and snapshot creation in Science Capsule. The snapshot manager tracks, saves and allows for temporal navigation (moving back and forth in time) of workflow executions by allowing users to create and retrieve snapshots. A 'workflow snapshot' is essentially a collection of data, metadata, initial state, derivation history, and current state of a workflow that that can be used to reproduce or revert to a workflow state. Currently, Science Capsule uses git as the version control back-end and stores object (data and code) deltas into a git database to create and manage snapshots. Corresponding metadata for the snapshots are stored in a relational (sqlite) database for efficient querying of snapshot information. The git storage back-end can be either an object database (Git-objectDB) or the git large file storage (Git-LFS). Since Git-LFS is designed for efficient tracking of

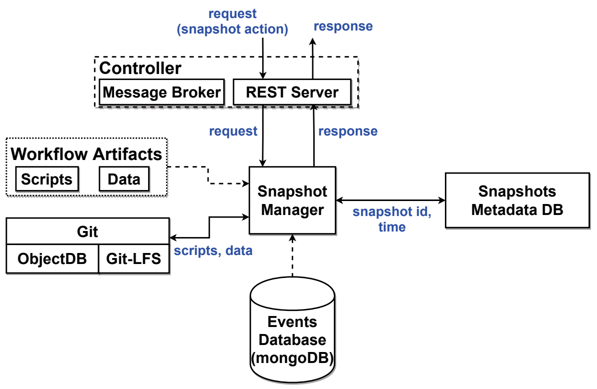


Fig. 3: Science Capsule tracks workflows and creates snapshots to save the state of a workflow and related artifacts. The snapshot manager uses Git to keep track of workflow scripts and data. When a user requests to create a workflow snapshot, the current state of the workflow is saved with a snapshot id. All workflow events are associated with the respective snapshot.

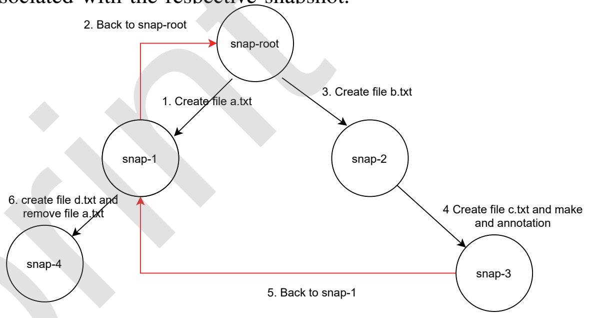


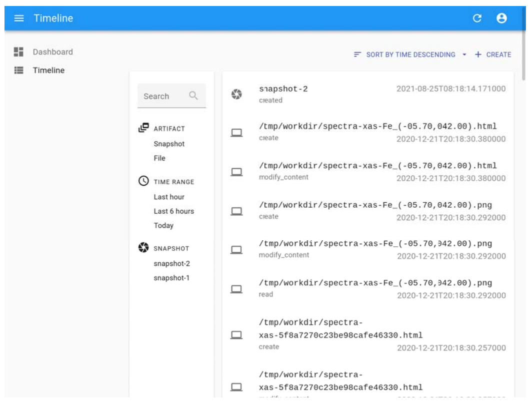
Fig. 4: Science Capsule snapshot example. Snapshotting in Science Capsule creates a tree structure where a node represents the state of a workflow during snapshot. The root of a snapshot tree is the first saved state of a workflow. Users can arbitrarily switch to a snapshot (red arrow) from another snapshot, resulting in multiple workflows or different versions of a workflow. Science Capsule captures the lineage of the workflows and their versions.

large files, Science Capsule by default uses Git-LFS and only falls back to Git-objectDB if the former is not available on the user's system. The lineage information for each snapshot is derived from the events captured by the monitoring tools in Science Capsule. Science Capsule creates an internal identifier to uniquely associate the events in a workflow to the specific snapshot.

Users can create multiple snapshots of a workflow over time, and can also roll back to specific snapshots for creating multiple execution traces of the workflow. This creates a tree of several snapshots (Figure 4), where each node of the tree refers to a version of the original workflow. The path to a node in the snapshot tree describes the lineage of the derived workflow as well as its artifacts. In addition to creating snapshots, users can also retrieve snapshots in any order, list snapshots, compare two snapshots, and find the objects and derivation history of snapshots.

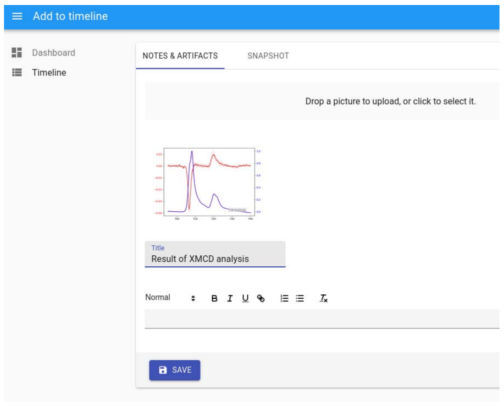
C. Science Capsule Implementation

Science Capsule is implemented in Python 3 and provides a command-line interface (CLI), a REST API, and a web-



```
snapshot-2                                                     2021-08-25T08:18:14.171000
created
/tmp/workdir/spectra-xas-Fe_(-05.70,042.00).html
create                                                         2020-12-21T20:18:30.380000
/tmp/workdir/spectra-xas-Fe_(-05.70,042.00).html
modify_content                                                 2020-12-21T20:18:30.380000
/tmp/workdir/spectra-xas-Fe_(-05.70,042.00).png
create                                                         2020-12-21T20:18:30.292000
/tmp/workdir/spectra-xas-Fe_(-05.70,342.00).png
modify_content                                                 2020-12-21T20:18:30.292000
/tmp/workdir/spectra-xas-Fe_(-05.70,342.00).png
read                                                           2020-12-21T20:18:30.292000
/tmp/workdir/spectra-
xas-5f8a7270c23be98cafe46330.html
create                                                         2020-12-21T20:18:30.257000
/tmp/workdir/spectra-
xas-5f8a7270c23be98cafe46330.html
```

(a) Science Capsule web client showing a workflow timeline



Title: Result of XMCD analysis

(b) Adding artifacts to workflow timeline

Fig. 5: Science Capsule web client displaying `XMCD-Spectral` workflow activities in a timeline view. Users can add images and/or notes to the timeline describing their workflow (beyond the compute or data life cycle).

based user interface. The CLI is used to start and manage the services (message broker, system monitors, database server etc.). The REST API is used to retrieve and analyze events that are captured and processed within Science Capsule, and also create and manage snapshots of workflows. The web client uses the API to retrieve and display the timeline of activities and artifacts of any workflow monitored by Science Capsule. Users can also integrate Science Capsule with existing workflow tools through the REST API.

The web client in Science Capsule displays, and allows users to analyze and annotate workflow and data life cycle events. This interface represents the events as timeline of workflow activities, visible in Figure 5. By default, the timeline shows all data and process events in a workflow as captured by Science Capsule. However, users can filter and sort activities based on their type, allowing them to analyze activities that are relevant to them. The web client is developed in React and is currently divided into two parts– a) the summary view (left side in Figure 5a) represents the types of artifacts and list of snapshots, and b) the right side on the web client shows the detailed information about the events, and organizes them in ascending/descending order of their occurrence. Each entry in the timeline shows relevant information about the artifacts (e.g, when a file has been written, its path, and the program invocation that generated the file). Figure 5b shows the interface to add notes or upload additional artifacts for the workflows (e.g., scans of lab notebooks), associating them with specific system events and/or time points in the workflow. This allows Science Capsule to record artifacts and activities beyond the computational timeline of a workflow.

We are also exploring integration of Science Capsule with the Jupyter ecosystem [31] through the development of a JupyterLab extension. This integration will allow users to run Science Capsule UIs as tabs in JupyterLab, and be able to capture, share and visualize the life cycle of workflows. This integration between Science Capsule with the Jupyter ecosystem will also allow scientific research and workflows managed through Jupyter to readily incorporate provenance collection and reproducibilty in their existing practices.

Finally, Science Capsule is well-integrated with current container technologies like Docker and Shifter. The Science Capsule CLI provides a wrapper over several docker commands to initiate and manage workflows using containers that have Science Capsule pre-installed and enabled. When Science Capsule services and workflows run inside a container, the monitoring system in Science Capsule captures all process and data events associated with the workflows, along with the underlying environment. This results in both data and process provenance, and hence, better ability to reproduce them. Additionally, when Science Capsule is used in a container, users can also share their workflows along with their artifacts. The Science Capsule CLI allows users to save and export workflows as container images that can be shared between users.

IV. Results

In this section, we evaluate the performance of Science Capsule using two scientific workflows and a synthetic benchmark across different computing environments.

A. Machines

We use a MacOS desktop (Darwin Kernel Version 19.3.0), a Linux server running Ubuntu v14.04.6, and the Cori supercomputer at NERSC for evaluating the different characteristics of Science Capsule. The Mac desktop has a 1.6 GHz Dual-Core Intel Core i5 processor with 16 GB of RAM. The Linux server has 35 GB of memory and 12 1600 MHz processors. Cori is a Cray XC40 supercomputer with 1630 compute nodes with 32 cores per node and 128 GB DDR4 2133 MHz memory. For our results from Cori, we used one compute node and the DRAM-based tmpfs file system with peak performance of approximately 1.3 GBps. We also evaluated Science Capsule using Docker (v17.05) containers on the Linux server, but only present results for the native Linux server as their performance was similar.

B. Workflows

In this section, we briefly describe the workflows used for evaluating Science Capsule.

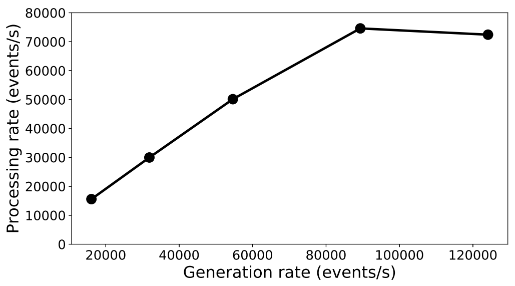


Fig. 6: The graph shows the rate at which Science Capsule can process inotify events generated by the `scbench` application on a Linux server. The processing rate increases linearly with the event generation rate up to 50,000 events/s. Due to the overheads of processing and transforming those events, the processing rate gets affected if more than 50,000 inotify events are generated. However, Science Capsule still achieves a maximum processing rate of 80,000 events/s.

`XMCD-Spectral` is a real-world workflow for online analysis of X-ray spectroscopy data at the Advanced Light Source (ALS). It has a Python CLI that allows users to load tabular data collected from a trajectory scan of a material sample. For our experiments, we used data from X-ray Magnetic Circular Dichroism (XMCD), and generated output files with spectral information as tables (CSV) and figures (PNG). For this workflow, Science Capsule monitored the directory where the input data was located and the output files were saved.

`Montage` is a data-intensive workflow that assembles an image from sky survey data (FITS files). It is a combination of sequential and parallel tasks. We used the python implementation of the workflow, and used different spatial scales to control the size of the images. `Montage` creates multiple sub-directories for saving and processing intermediate files. Science Capsule was configured to monitor all the directories and sub-directories created and accessed by `Montage`.

`scbench` is a synthetic benchmark that we developed for evaluating Science Capsule. `scbench` allows us to control the data (and subsequently, event) generation rate, number of files and data size. We used `scbench` to create and write 4 KB files in parallel for a fixed duration to measure different performance implications in Science Capsule.

C. Processing Rate

Figure 6 shows the rate at which Science Capsule captures, filters and aggregates raw, unprocessed events from different monitoring sources on the Linux server. For this experiment, we used `scbench` to do a fixed number of I/O operations on files and directories monitored by Science Capsule, thus controlling the event generation rate through `scbench`. The result shows that Science Capsule could process $\approx$ 50,000 events/s on par with the event generation rate. Additionally, the throughput performance goes up to a maximum of 80,000 events/s, but with some backpressure[*]. However, real-world workflows often generate events at a much lower rate – `Montage` generates $\approx$ 5K events/s for spatial scale 3.0, and `XMCD-Spectral` generates $\approx$ 80 events/s on the Linux server – than can be processed by Science Capsule in (near) real-time.

[*]Condition where event generation rate is higher than the event processing rate, resulting in continuously increasing the number of unprocessed events in the queue.

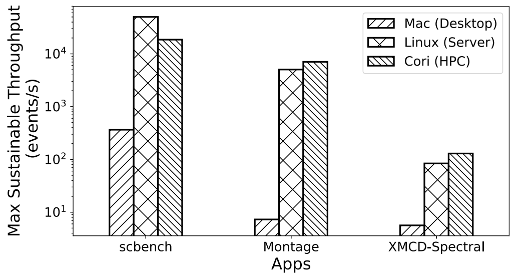


Fig. 7: Maximum sustainable throughput of Science Capsule for processing events depends on the monitoring source, underlying platform and the applications' throughput. For real-world workflows (`XMCD-Spectral` and `Montage`), throughput is higher on Cori as data is processed faster. The throughput of `scbench` on Cori is limited by the queue size of inotify, which can be updated by a system administrator to handle large event rates.

Figure 7 shows the maximum sustainable throughput obtained by Science Capsule for the workflows on different platforms. On the Mac desktop, the processing rate is low because the rate at which the events are generated is low. Watchdog implicitly aggregates raw system-level events, and generates significantly fewer events as compared to inotify or strace. Additionally for `Montage` and `XMCD-Spectral`, event generation is higher on Cori than on other systems as the workflows run faster and generates more events/s. For `scbench`, the event processing rate is the maximum when the application runs on the Linux server. On Cori, the processing rate is low because of the small queue size in inotify that limits the maximum number of events in the queue, and needs to be updated by an admin. The results show that Science Capsule throughput depends on the monitoring source, underlying platform and the throughput at which workflow events are generated.

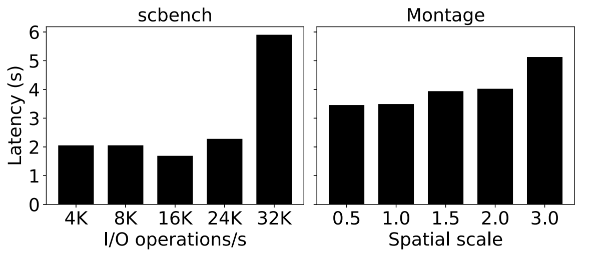
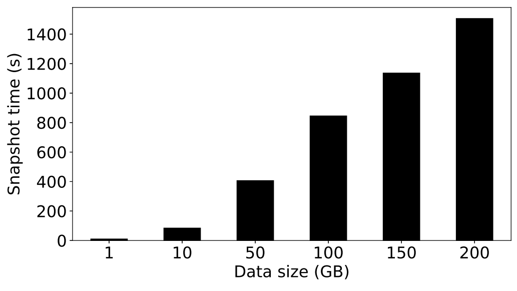


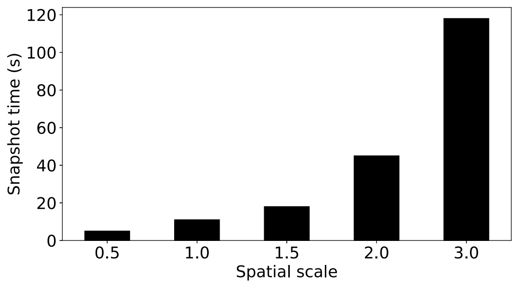
Fig. 8: Latency depends on the overheads of processing the raw events in Science Capsule. The latency increases when the event generation rate exceeds the event processing rate due to large number of I/O operations/s.

D. Processing Latency and Overhead

Figure 8 shows the processing latency in Science Capsule, which measures the time between the last generated unpro-



(a) scbench



(b) Montage

Fig. 9: Time to create snapshots using Science Capsule for `scbench` and `Montage` workflows (`XMCD-Spectral` had similar patterns) generating different data sizes and number of files. For `Montage`, total data size and the number of files increase as the spatial scale increases. Snapshot time increases with increasing data size, which can be minimized by creating snapshots in the background or incremental snapshots whenever possible.

App	Raw events	Agg. events	% Reduction
`scbench`	1024008	16	99.998
`Montage`	4608170	17596	99.6
`XMCD-Spectral`	1724	90	94.8

TABLE I: Science Capsule processes raw, system events into reduced, aggregated events based on the number of files and a time window. `scbench` and `XMCD-Spectral` work with few files, resulting in a small number of processed events. `Montage` operates on many files, resulting in fairly large number of processed events.

cessed event and the last processed event. The workflows are executed on the Linux server for this experiment. We do not plot the latency for `XMCD-Spectral` since the event generation rate is low ($\approx$ 80 events/s), and Science Capsule obtains sub-second latency. For both `scbench` and `Montage`, the latency is minimal (<6s) for large number of I/O operations and files. For `scbench`, when the number of I/O operations per second is less than 24,000 Science Capsule maintains a processing latency of $\leq$3s because it can process at almost the same rate at which the events are generated. When the number of I/O operations per second exceed beyond 24,000, the latency increases because of the high event generation rate (>50,000 events/s; Section IV-C shows that backpressure occurs when the event generation rate exceeds beyond 50,000).

We also studied the computation overhead of using Science Capsule. Since by design, Science Capsule does not actively intercept the application execution, and collects events from system-level monitoring logs, the overheads on workflow performance were minimal. Our experiments show that for the workflows evaluated in this paper, the execution overhead does not exceed beyond 5s, specifically on platforms that are constrained by CPU and memory resources (e.g., Mac desktop).

E. Implications of Event Aggregation

Table I shows the reduction in the number of events after Science Capsule aggregates the unprocessed monitoring events generated during workflow execution. In this result, we only report the number of events generated by the inotify monitor as the aggregation and reduction is significantly more important for inotify, than any other monitoring sources (e.g., watchdog) due to its fine granularity of capturing the system events. The table shows that for `Montage` and `XMCD-Spectral`, the reduced number of events is $\approx$ 0.4% and 5% respectively. Science Capsule aggregates events based on a time window and the number of files. In case a file has been accessed and used multiple times within the same time window, Science Capsule creates a single event with aggregated statistics instead of a large number of system events that are originally generated by the event monitors. For `scbench`, the number of aggregated events is significantly small ($< 0.001\%$) as compared to the number of unprocessed events because `scbench` does multiple write operations on a small number of files that were grouped as a single write event per file.

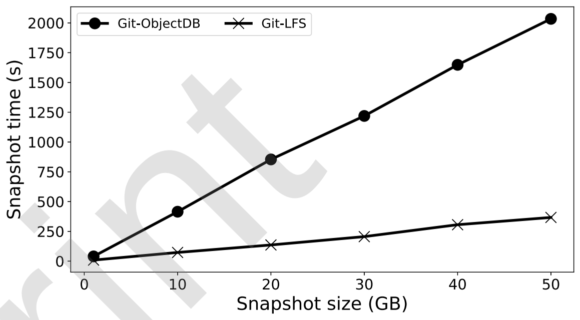


Fig. 10: Time to create snapshots of the `scbench` workflow (including data) using different backends. Snapshotting with Git-LFS backend performs better than Git ObjectDB as Git-LFS handles large files efficiently.

F. Snapshot Performance and Overheads

In this section, we present our evaluation for the snapshot feature in Science Capsule, which allows users to save, reuse and share their workflow state (including data). We evaluate the different snapshot actions, effect of different storage backends, and performance overheads for different data sizes. The results use git-lfs as the backend unless otherwise specified.

Snapshot create time. Figure 9 shows that the time to create snapshots using Science Capsule increases with increasing data size and number of files. Figure 9a shows that the time to create snapshots increase linearly with increasing data size for `scbench`. Figure 9b shows the time to create snapshots increase for the `Montage` workflow with increasing values of the spatial scale. With larger values of spatial scale, `Montage` generates more files, resulting in both increasing number of files and data size to snapshot. However for large snapshots, the time can be minimized by creating them in the background or choosing incremental snapshots of small size.

Effect of different storage backends. Figure 10 shows the performance of creating snapshots using the two git backends – git-lfs and git-objectDB – in Science Capsule. The graph shows that with the git-lfs backend, Science Capsule creates snapshots significantly faster ($\approx 5\times$) than using the default objectDB for larger data sizes. The results imply that users can efficiently create and manage large snapshots of their workflows and data using Science Capsule.

V. Conclusions

Science Capsule captures the processing and human elements of increasingly complex data life cycles that are necessary to share, reuse and reproduce workflows across heterogeneous computing environments. Science Capsule actively monitors workflows, processes workflow events in real-time to generate provenance, and allows users to explicitly enhance and analyze collected provenance information. Science Capsule reduces the user burden in collecting, aggregating, and viewing workflow details from user notes to system events. Subsequently, users can use the provenance journal to reconstruct/reproduce their workflows. However, Science Capsule does not guarantee complete provenance and it is possible that in the process of sharing or reproducing, users might not have a complete workflow which they may choose to augment. Our evaluation shows that Science Capsule provides a high-throughput, low-latency framework to capture the relevant information without any significant performance overheads.

In future work, we will explore ways to extract relevant events and provenance information from workflow logs that will simplify integrating existing workflows with Science Capsule. User interaction and appropriate abstractions for interacting with collected provenance data will also be explored in future work.

Acknowledgments

This work and the NERSC resources used in the project are supported by the U.S. Department of Energy, Office of Science and Office of Advanced Scientific Computing Research (ASCR) under Contract No. DE-AC02-05CH11231.

References

[1] I. J. Taylor, E. Deelman, D. B. Gannon, and M. Shields, *Workflows for e-Science: scientific workflows for grids*. Springer, 2014.

[2] E. Deelman *et al.*, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," *Scientific Programming*, vol. 13, no. 3, pp. 219–237, 2005.

[3] A. Jain *et al.*, "Fireworks: A dynamic workflow system designed for high-throughput applications," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 17, pp. 5037–5059, 2015.

[4] R. J. Pandolfi *et al.*, "Xi-cam: a versatile interface for data visualization and analysis," *Journal of synchrotron radiation*, vol. 25, no. 4, pp. 1261–1270, 2018.

[5] I. Altintas, O. Barney, and E. Jaeger-Frank, "Provenance collection support in the kepler scientific workflow system," in *International Provenance and Annotation Workshop*. Springer, 2006, pp. 118–132.

[6] R. S. Barga and L. A. Digiampietri, "Automatic generation of workflow provenance," in *International Provenance and Annotation Workshop*. Springer, 2006, pp. 1–9.

[7] F. Chirigati, R. Rampin, D. Shasha, and J. Freire, "Reprozip: Computational reproducibility with ease," in *Proceedings of the 2016 international conference on management of data*, 2016, pp. 2085–2088.

[8] A. Brinckman *et al.*, "Computing environments for reproducibility: Capturing the "whole tale"," *Future Generation Computer Systems*, vol. 94, pp. 854–867, 2019.

[9] T. Šimko, L. Heinrich, H. Hirvonsalo, D. Kousidis, and D. Rodríguez, "Reana: A system for reusable research data analyses," in *EPJ Web of Conferences*, vol. 214. EDP Sciences, 2019, p. 06034.

[10] P. J. Guo and M. I. Seltzer, "Burrito: Wrapping your lab notebook in computational infrastructure," 2012.

[11] P. Alper, K. Belhajjame, V. Curcin, and C. A. Goble, "Labelflow framework for annotating workflow provenance," in *Informatics*, vol. 5, no. 1. Multidisciplinary Digital Publishing Institute, 2018, p. 11.

[12] J. Cała and P. Missier, "Provenance annotation and analysis to support process re-computation," in *International Provenance and Annotation Workshop*. Springer, 2018, pp. 3–15.

[13] A. Rajasekar *et al.*, "iRODS Primer: Integrated Rule-Oriented Data System," *Synthesis Lectures on Information Concepts, Retrieval, and Services*, vol. 2, no. 1, pp. 1–143, 2010.

[14] J. M. Hellerstein *et al.*, "Ground: A data context service."

[15] A. Arkilic *et al.*, "Databroker: An interface for nsls-ii data management system," in *15th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'15), Melbourne, Australia, 17-23 October 2015*. JACOW, Geneva, Switzerland, 2015, pp. 645–647.

[16] J. P. Birnholtz and M. J. Bietz, "Data at work: Supporting sharing in science and engineering," in *Proceedings of the 2003 International ACM SIGGROUP Conference on Supporting Group Work*. New York, NY, USA: Association for Computing Machinery, 2003, p. 339–348.

[17] J. C. Wallis, E. Rolando, and C. L. Borgman, "If we share data, will anyone use them? data sharing and reuse in the long tail of science and technology," *PLoS ONE*, vol. 8, no. 7, p. e67332, 2013.

[18] M. Konkol, D. Nüst, and L. Goulier, "Publishing computational research - a review of infrastructures for reproducible and transparent scholarly communication," *Research Integrity and Peer Review*, vol. 5, no. 1, 2020.

[19] X. Chen *et al.*, "Open is not enough," *Nature Physics*, vol. 15, no. 2, pp. 113–119, 2019.

[20] V. Stodden, J. Seiler, and Z. Ma, "An empirical analysis of journal policy effectiveness for computational reproducibility," *Proceedings of the National Academy of Sciences*, vol. 115, no. 11, pp. 2584–2589, 2018.

[21] J. Zhao, C. Goble, R. Stevens, and D. Turi, "Mining taverna's semantic web of provenance," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 5, pp. 463–472, 2008.

[22] P. Chen, B. Plale, and M. S. Aktas, "Temporal representation for mining scientific data provenance," *Future generation computer systems*, vol. 36, pp. 363–378, 2014.

[23] D. Fensel, *Ontologies: A silver bullet for knowledge management and electronic commerce*. Springer Science & Business Media, 2003.

[24] M. Uschold and M. Gruninger, "Ontologies: Principles, methods and applications," *The knowledge engineering review*, vol. 11, no. 02, pp. 93–136, 1996.

[25] P. Missier, K. Belhajjame, and J. Cheney, "The w3c prov family of specifications for modelling provenance metadata," in *Proceedings of the 16th International Conference on Extending Database Technology*, 2013, pp. 773–776.

[26] Z. Bao, S. Cohen-Boulakia, S. B. Davidson, and P. Girard, "Pdiffview: Viewing the difference in provenance of workflow results," *Proceedings of the VLDB Endowment*, vol. 2, no. 2, pp. 1638–1641, 2009.

[27] P. Missier, S. Woodman, H. Hiden, and P. Watson, "Provenance and data differencing for workflow reproducibility analysis," *Concurrency and Computation: Practice and Experience*, vol. 28, no. 4, pp. 995–1015, 2016.

[28] J. Freire and C. Silva, "Simplifying the design of workflows for large-scale data exploration and visualization," in *Proceedings of the Microsoft eScience Workshop*, 2008.

[29] R. Love, "Kernel korner: Intro to inotify," *Linux Journal*, vol. 2005, no. 139, p. 8, 2005.

[30] "watchdog," https://pypi.org/project/watchdog/.

[31] Project Jupyter, "Project Jupyter Home," https://jupyter.org/, 2021.