

Computational reproducibility of scientific workflows at extreme scales

Line Pouchard¹ , Sterling Baldwin², Todd Elsethagen³, Shantenu Jha¹, Bibi Raju³, Eric Stephan³, Li Tang⁴ and Kerstin Kleese Van Dam¹

The International Journal of High Performance Computing Applications 2019, Vol. 33(5) 763–776

© The Author(s) 2019

Article reuse guidelines:

sagepub.com/journals-permissions

DOI: 10.1177/1094342019839124

journals.sagepub.com/home/hpc



Abstract

We propose an approach for improved reproducibility that includes capturing and relating provenance characteristics and performance metrics. We discuss two use cases: scientific reproducibility of results in the Energy Exascale Earth System Model (E3SM—previously ACME) and performance reproducibility in molecular dynamics workflows on HPC platforms. To capture and persist the provenance and performance data of these workflows, we have designed and developed the Chimbuko and ProvEn frameworks. Chimbuko captures provenance and enables detailed single workflow performance analysis. ProvEn is a hybrid, queryable system for storing and analyzing the provenance and performance metrics of multiple runs in workflow performance analysis campaigns. Workflow provenance and performance data output from Chimbuko can be visualized in a dynamic, multilevel visualization providing overview and zoom-in capabilities for areas of interest. Provenance and related performance data ingested into ProvEn is queryable and can be used to reproduce runs. Our provenance-based approach highlights challenges in extracting information and gaps in the information collected. It is agnostic to the type of provenance data it captures so that both the reproducibility of scientific results and that of performance can be explored with our tools.

Keywords

Computational reproducibility, scientific workflows, provenance, performance analysis, ProvEn, Chimbuko

1. Introduction

The computational reproducibility of scientific experiments performed in diverse computing environments is both critical for the credibility of scientific results and difficult to achieve as numerous factors simultaneously impact the goals of reproducibility. As workflows are more widely adopted in operational settings such as real-time experimental and computational data analysis, a secondary challenge is performance reproducibility,—can we reliably deliver the required results in a given tight time window or resource constraint? Both reproducibility challenges are strongly influenced by their computational conditions that can refer to a litany of factors including: computer architecture, system software, reference data, application configuration, and third-party libraries. In particular, the Department of Energy (DOE) Leadership Class Facilities (LCFs) exhibit an increasingly heterogeneous hardware environment, on which multiple versions of quickly evolving, unique libraries are made available. Against the backdrop of complicated architectures with large heterogeneity of devices, complex memory hierarchies, and complex communication patterns, teams face an increasing need to

establish credibility by producing reproducible science.

Definitions of computational reproducibility vary largely and depend on context with several, sometimes complementary, sometimes orthogonal axes: scientific replicability of results to validate discoveries, numerical reproducibility to validate code correctness, and performance reproducibility during time or resource constraint executions are all relevant to scientific computing. The taxonomy of computational reproducibility is explained by Heroux et al. (2018), and it differentiates reproducibility achieved by different teams on different experimental setups from reproducibility achieved by different teams on the same experimental setup. The taxonomy highlights these

¹ Brookhaven National Laboratory, Upton, NY, USA

² Lawrence Livermore National Laboratory, Livermore, CA, USA

³ Pacific Northwest National Laboratory, Richland, WA, USA

⁴ Los Alamos National Laboratory, Los Alamos, NM, USA

Corresponding author:

Line Pouchard, Brookhaven National Laboratory, PO Box 5000, Upton, NY 11973, USA.

Email: pouchard@bnl.gov

two broad definitional themes and names them reproducibility and replicability. These labels are swapped between the recent ACM efforts and the historical Clearbout theme. Reproducibility of performance is not addressed in this taxonomy, and we introduce this aspect.

Reproducibility goals and methods to attain these goals will vary depending on scientific applications and the communities around them. In some cases, reproducibility goals are implicit, with little formally expressed consensus on what is considered reproducibility and what methods should be used to achieve these goals. Typically motivations for improving reproducibility of a scientific experiment address a fundamental tenet of the scientific method. They may also include the need for obtaining consensual results from which policy is derived, as in climate research, and the ability of supporting authors' claims that their methods or algorithms represent a progress over previous work. With the increased complexity of heterogeneous architectures and potential for lack of reproducible accuracy, it will fall on each scientific community to define its own reproducibility goals and error bounds.

In a parallel environment, bitwise reproducibility is used during specific development phases, such as with the Energy Exascale Earth System Model (E3SM—previously ACME) project (Koch et al., 2016), the first use case is presented here. Bitwise reproducibility is defined as the ability to recreate a run and have its output match that of a previous run bit-for-bit. Bitwise reproducibility of scientific results is important for climate modeling applications where scientific code is developed by multiple, distributed teams. Versions of the codes must provide bitwise identical results when using observational input data to ensure code stabilization during a code development phase. When the code is considered stable, this constraint can be relaxed. For applications running in parallel, achieving bitwise reproducibility throughout would require putting severe constraints on the order of execution for instructions and operations in distributed data structures: other definitions of reproducibility are needed.

A workflow execution can exhibit performance fluctuations due to external factors such as issues of contention for shared system resources, including contention with itself, the order in which memory is accessed on multicore systems, i/o, communication latencies. We present workflow performance fluctuations as a second case illustrating reproducibility. Given the resource limitations and the need to complete simulations in a manageable time period, the characterization of performance in heterogeneous architectures becomes a key component of reproducibility, because performance gains can imply potential trade-offs in achievable levels of reproducibility for a given domain application. If simulations are fundamentally stochastic and have an additional contribution to the random variations in their time-to-completion, “full blanket” reproducibility will be difficult to achieve and is best viewed as a multiple level capability characterized by different trade-offs and costs. Performance fluctuations will exist for the same codes on

the same and on different architectures and resource utilization on shared systems will also impact performance. The following questions become necessary to address: Under what runtime conditions is reproducibility achievable? Should performance considerations be part of a definition of reproducibility? The numerous factors influencing performance reproducibility argue for capturing and persisting detailed provenance information for each run in reproducibility campaigns so that during code optimization application developers can compare performance runs in their execution context. Reproducibility implies a critical requirement to record the conditions under which experiments are performed.

We propose an approach for improved reproducibility based on provenance that captures the parameters of the experiment and computing environments used by computational workflows and relates the provenance characteristics to performance metrics where needed. The goal is to extract the pertinent information, convert it into a unified format, and provide easy access, thus enabling reproducibility campaigns. To illustrate the challenges envisioned with achieving reproducibility at the exascale and beyond, we present two use cases and we discuss the heterogeneity of relevant information, the need for formalized computational workflows, and the adaptive nature of simulations. To capture and persist the provenance and performance data of these workflows, we have designed and developed the Chimbuko¹ and Provenance Environment (ProvEn)² frameworks. Chimbuko captures provenance and enables detailed single workflow performance analysis. ProvEn is a hybrid, queryable system for storing and analyzing the provenance and performance metrics of multiple runs in workflow performance analysis campaigns. Our provenance-based approach highlights challenges in extracting information and gaps in the information collected. It is agnostic to the type of provenance data that it captures so that both the reproducibility of scientific results and that of performance can be explored with our tools.

2. Use case 1: Reproducibility of scientific results

Global climate models have a unique position in the physics simulation world in that they exist to both answer scientific questions and inform public policy. This coupled with the extremely large and complex nature of its subject matter means that reproducibility is absolutely critical to the success of the E3SM project. There are two main driving factors behind the E3SM project's needs. First, all data produced by the model must be independently verifiable or it loses credibility. Second, the E3SM development team is widely distributed across many research institutions and heterogeneous compute facilities. As the project develops and the code progresses, it's becoming increasingly apparent that every step to produce a piece of data absolutely must be accessible when viewing the model output. Due to the extremely computationally intensive nature of

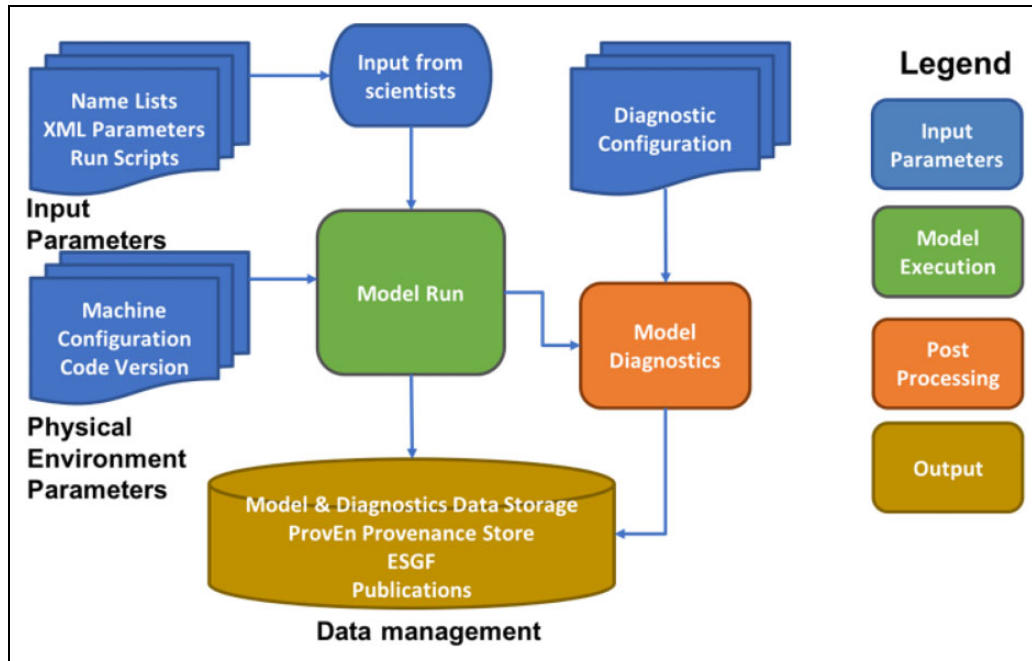


Figure 1. E3SM end-to-end workflow. E3SM: Energy Exascale Earth System Model; ESGF: Earth Systems Grid Federation.

producing model output, careful consideration must be made when setting up a new run, which means previous successful run configurations greatly influence choices for future simulations. Historically, much of this information has been captured manually and made available via shared documents. But as both the project team and the code base grow, this manual approach increasingly fails to robustly capture data critical for reproducibility. When collecting run configuration data manually, member teams working on different submodules will collect different information, leaving us with irregular and incomplete data. Completely reproducing a run not only requires the input parameters, such as surface temperature forcing, but also machine specifications such as compiler flags, environment variables, number of cores, as well as the git hash of the code used.

Figure 1 describes a simplified version of the E3SM end-to-end workflows. E3SM requires several types of input (depicted in blue boxes) that must all be persisted to achieve the goal of bitwise reproducibility. The E3SM model has a set of default values captured in Name Lists and Run Scripts that scientists can change based on their experimental design to study specific physical interactions in the model. For example, someone interested in how the model responds to an increase in short wave radiation captured in the upper atmosphere can change the input Name Lists to increase heat above a given altitude. Input from scientists also include Initialization Files and Input data sets. Physical environment parameters include machine configuration and drive the model run on systems such as the LCFs. These include parameters that can change for each run such as the number of nodes used for a given run and characteristics of the system itself, such as how many cores per node and how much RAM per core is available.

The code version for each run must be recorded because the software engineering team optimizes the simulation code to take advantage of the architecture of the HPC facility where the codes are run. Diagnostic configuration parameters are used in running diagnostics and analysis such as UV-CDAT (orange box) to verify model output. Diagnostics needs 5, 10, or 20 years of history files to produce climatology files so this analysis can happen at locations with larger storage allocations and other than where the model is run. They include, for instance, the observation sets the results are being compared to and the plots to be produced. As bitwise reproducibility is the goal for E3SM, capturing all the input parameters, configurations, and system variables for both model run and diagnostics is crucial.

The Data Management layer (tan box) has the task to store result data and their provenance over the long-term (decades) as well as making it accessible to scientists studying climate worldwide. It includes many options, such as the E3SM database and the Earth Systems Grid Federation (ESGF) (Cinquini et al., 2014). The ProvEn framework persists and allows provenance queries to be executed on all input parameters used in running E3SM, including diagnostic inputs and which models are run by geographically distributed teams.

3. Use case 2: reproducibility of performance

As part of our research into and the development of “building blocks” for workflow systems (Turilli et al., 2016), we recently investigated the performance reproducibility of two distinct workflow systems at progressively increasing scales. These workflows were drawn from

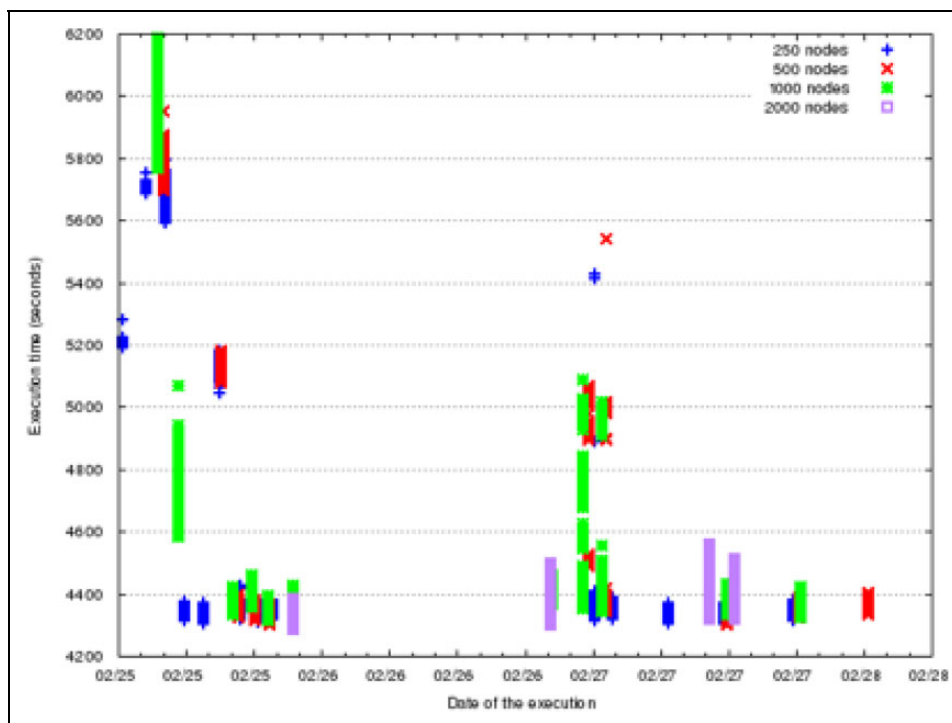


Figure 2. Performance variations of N concurrent MD tasks on varying number of nodes. MD: molecular dynamic.

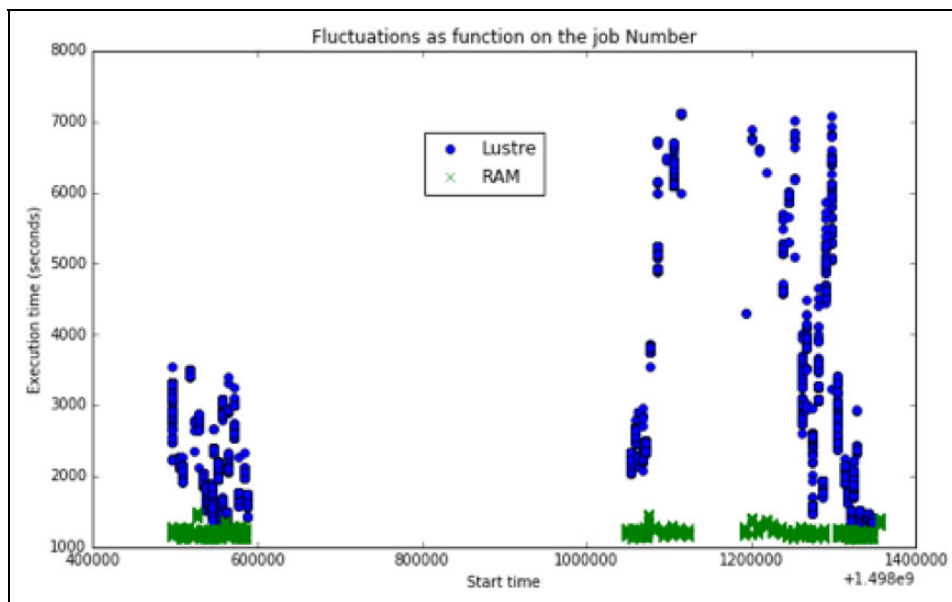


Figure 3. Performance variations of N concurrent AthenaMP tasks.

biomolecular simulations and high-energy physics and were executed on DOE LCFs. The first use case is comprised of multiple concurrent executions of molecular dynamic (MD) simulations using the Gromacs MD engine (Figure 2); the high-energy physics use case is constituted of multiple AthenaMP tasks (where each task runs on one node; Figure 3).

Significant fluctuations in the total time-to-completion (TTC) between different runs of the same workflow were

observed. The workflows were by design comprised of identical tasks, and thus the time to completion of each task should in principle have been identical. Contrast these with more traditional workflows that are typically comprised of heterogeneous tasks, have complex dependencies between them and are typically represented by Directed Acyclic Graphs. However, the individual and independent tasks of both workflows—biomolecular and high-energy physics—had different execution times (T_x). The fluctuations in T_x

were independent of the scale of the workflow (as measured by number of tasks) over two orders of magnitude ($O(10)$ – $O(1000)$ tasks) and number of codes. Multiple identical workflows were repeated over a duration of many weeks; the fluctuations in the T_x on individual tasks were however statistically indistinguishable across repeats. Figures 2 and 3 represent fluctuations over time (i.e. multiple repeats) and scale of the workflow.

Figure 2 plots the fluctuations in the T_x of N concurrent MD tasks, where N varies from 250 to 2000. Figure 3 plots the fluctuations in T_x of N concurrent AthenaMP tasks, when the filesystem is used (blue) and when the filesystem (Lustre) isn't used (green). Although the usage of the filesystem clearly contributes to the fluctuation of T_x , it is not the only contribution. In addition to performance fluctuation, there are systematic errors in measurements. The primary contributions to systematic errors are due to inaccuracies in T_x measurements, which arise due to at least two reasons: (i) a lag between events occurring and time stamps associated with those events being recorded, (ii) coarse grained representation of the discrete events and transitions. However, the systematic errors are expected to be unbiased across different workflow experiments and runs.

4. Approach

Traditionally, workflows have only captured information about the workflow itself but not about the computing infrastructure and environment they are executed in. We contend that this additional information is vital if we want to achieve reproducibility and obtain a deeper understanding of the barriers to reproducibility, in particular in extreme scale computing environments. Our approach to improve reproducibility is to capture both provenance characteristics and performance metrics about the workflow, the computing infrastructure and the execution environment. We relate this information to obtain and persist detailed, queryable information useful for reproducibility, performance debugging, and code development and optimization. Our approach is encapsulated in the Chimbuko and ProvEn systems (Elsethagen et al., 2016; Pouchard et al., 2017).

4.1. Chimbuko

Real-time monitoring of scientific workflow processes is needed to validate performance and support reproducibility. Defining the appropriate level of abstraction for monitoring workflows is a challenge given the number of components, the complexity of the connections, and the rate of execution for each component. Changing computer architectures, computational conditions, system environments, applications and runtime configurations, and third-party libraries need to be recorded, persisted, and accessible to support forensic investigations of performance fluctuations or differences in the scientific results.

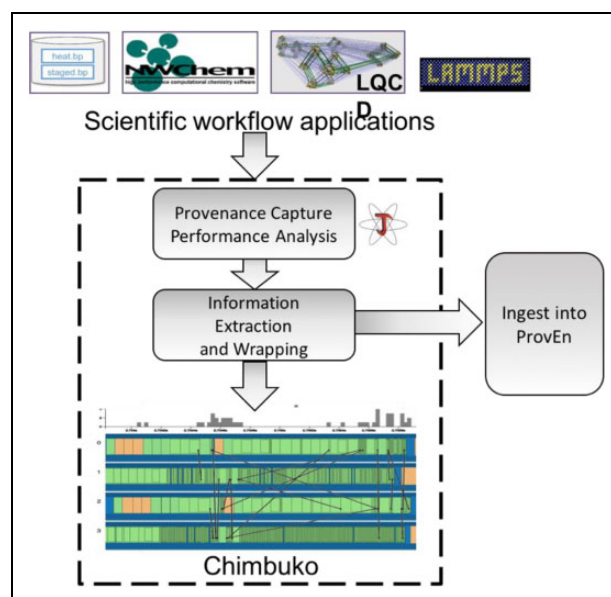


Figure 4. Chimbuko framework and its relationship to ProvEn. ProvEn: Provenance Environment; LAMMPS: large-scale atomic/molecular massively parallel simulator.

The Chimbuko framework systematically captures, stores and correlates provenance and performance metrics of workflows and systems, and offers the ability to analyze performance over time and to plug in various third party performance tools, such as the Tuning and Analysis Utilities (TAU) (Huck et al., 2006). Workflow provenance and performance data output from Chimbuko can be visualized in a dynamic, multilevel visualization providing overview and zoom-in capabilities for areas of interest (Xie et al., 2018). Chimbuko has been tested on workflow applications for the DOE Exascale Computing Program (ECP) such as NWChemEX (Valiev et al., 2010), LAMMPS and other MD codes. Figure 4 shows the components of Chimbuko.

Chimbuko currently uses TAU to extract performance information. TAU libraries have been modified to include the ability to collect performance information from workflows rather than just single applications: it hereby aggregates and integrates the information of all individual workflow components into one combined performance profile. In addition, TAU has been modified to provide provenance metadata by capturing configuration and environment parameters in addition to the performance metrics it natively extracts. Chimbuko uses the hierarchical taxonomy proposed in the Workflow Performance Provenance Ontology (Kleese van Dam et al., 2015) to specify the information to be extracted and customizes it for a particular application, as relevant metrics and provenance vary by application. Details on information extraction are provided in the next section. Chimbuko users compile their application with TAU 2.27 or later, using compiler or source code instrumentation, specify the output as profile (statistical summary information) or trace. Trace metrics is an event stream recording time spent in functions per node.

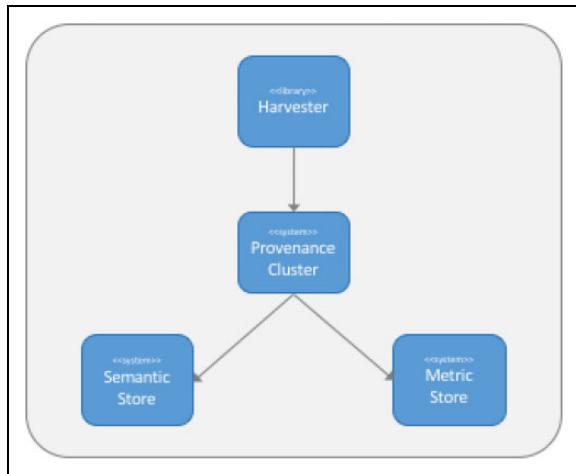


Figure 5. ProvEn architecture. ProvEn: Provenance Environment.

Provenance information is recorded by TAU and stored into Chimbuko, providing a way to investigate application behavior over time. Traces or profiles are analyzed and visualized to highlight potential bottlenecks. ProvEn provides the search capability over profile files.

4.2. ProvEn

The ProvEn extends classical provenance concepts to incorporate metrics, handle scalability, and undertake analytics to support large-scale science workflow performance optimizations. Classical provenance concepts exemplified in ProvEn include Provenance Creation through disclosure or harvesting; Provenance Capture for persistence into a provenance data store (e.g. a graph database); and Provenance Exploration using semantic graph query language to explore and reason over existing provenance data sets. ProvEn extends these concepts with the integration of time series data capture (metrics) linked to provenance data and an in-memory compute platform to provide both scalability and support for real-time analytics over large-scale scientific workflow applications. The ProvEn framework is composed of a Provenance Cluster, a Semantic Store, a Metric Store, and ingest mechanisms including a Harvester (Figure 5).

The Cluster is a partitioned In-Memory Data Grid (IMDG) and is scale-out capable for high throughput, distribution, and high availability. The cluster stores a subset of the collected data in memory for real-time analytics and reporting. All provenance and performance data are redirected to the Semantic and Metric stores for long-term storage. ProvEn's Provenance Cluster contains homogeneous member nodes, each of which is a Java-based Web module application hosted by a Payara Micro Java EE application container (Pilgrim, 2015). Member nodes are arranged in a masterless peer-to-peer network, with each node providing an identical set of services to capture and store, monitor, and query the incoming data streams. As data streaming activity increases or decreases new member nodes will join

or be removed respectively. Member nodes contribute their memory to the Cluster creating the partitioned IMDG and managed by the open-source Hazelcast (Johns, 2015). Streaming data are stored in the IMDG for fast “real-time” access, query, and analysis capabilities. The amount of data stored in the IMDG is of course limited, and its size is managed to not exceed the memory resources being contributed by member nodes. Data are moved from the IMDG to persistent stores (i.e. Semantic and Metric Stores) for long-term storage and to avoid out of memory issues.

Provenance data are disclosed to the Provenance Cluster by various methods, including the ProvEn's Harvester library, a push/pull method capable of crawling and enriching tabular data. Additional disclosure strategies include an embedded client java API for describing provenance, and a REST API using a form of Javascript Object Notation (JSON) for Linked Data (JSON-LD). Provenance harvesting is performed on identified source artifacts (e.g. log files, workflow configuration files, and file output from Chimbuko, including TAU profiling and trace files). ProvEn also supports post-run analytics and reporting with Jupyter and enables semantic queries. System metrics (e.g. CPU metrics, i/o metrics, memory metrics, etc.) is either collected out-of-band from provenance collection, providing contextual information for running applications, or provided by Chimbuko. System metrics together with disclosed provenance data contribute to explanation and verification of performance and application results. Provenance metrics provide a temporal link between disclosed provenance data and the observed system metrics.

5. Information extraction and capture

The aim of provenance for science applications is to provide a thorough explanation of events and resulting data with the goal of attaining a set level of reproducibility of scientific results and workflow performance. This can be typically achieved by disclosing historical information logged by a parent application or workflow listener. Complex distributed applications such as multi-tier workflows, distributed data management systems, and grid computing analytics require a different approach to collecting and integrating provenance to provide a composite understanding of the entire application. Precise provenance disclosure is not always possible in science applications and computing environments, due, for example, to security barriers and overhead costs. Fortunately, many science application developers have begun providing simplified forms of provenance information known as scruffy provenance (Moreau and Groth, 2013). This information is provided through debug logs, performance status indicators, configuration files, and performance audits that are largely represented in locations (compute nodes and directory trees) and tabular and parameter list formats providing relatable data structures, that are both human readable and computable. This simplified provenance information provides benefit largely to the user running science applications, because it is written using local terminology.



Figure 6. Details of metrics and provenance extraction.

Embedded throughout the application results directories scruffy provenance provides indicators on success or anomalies for a particular run. In the same token, the convenience of being embedded in application results make scruffy provenance harder to share among collaborators. Despite some limitations of scruffy provenance, the terminology is generally understood by application developers and consumers of the application results making it a good starting point for provenance enrichment.

Chimbuko extracts a combination of scruffy provenance and performance metrics such as performance traces and profiles. Trace files contain detailed performance monitoring, such as number, volume, and execution time of individual communication calls but also incur large storage overhead. The profiles extracted with Chimbuko contain summary statistics about communication and execution on cores; additional metrics such as the total execution time of a workflow are also extracted. Provenance information such as application names, versions, job configurations, versions of library used, and details about system architecture are extracted and stored in Chimbuko and queried with ProvEn. Figure 6 shows details of the information extracted by Chimbuko in a JSON script.

In Figure 6, workflow component description, metadata, and units are available for each workflow component. The timestamps are in microseconds. Total_time is the total execution time of a workflow (end-to-end workflow latency); aggr_communication_calls, _time, and _collective_bytes are the number, execution time, and volume of MPI calls; aggr_adios_bytes is the total I/O volume, here the ADIOS I/O [Version 1.0] data reduction library was used for this example (Liu et al., 2014). Under ./tau_data metadata, scruffy provenance for this run are persisted.

The focus of information extraction in ProvEn is to develop techniques to enrich scruffy provenance information through a lightweight and generalized syntax so that a more complete set of provenance data will be portable, easily shareable, and suitable for analysis even for large-scale applications. Using scruffy provenance as the starting point, enriching provenance information with syntax at harvest time provides the added benefit of conveying provenance concepts based on domain terminology to help retain and reinforce similar mental models relatable to the model developers and consumers.

6. Use case studies and discussion

6.1. Scientific reproducibility use case—E3SM

In this initial case, reproducibility is performed in two stages we call bundling and enrichment. The first stage, bundling, involves collecting key elements extracted from an existing E3SM simulation run and ingesting a subset of user modified files retained in their native form. The key elements identified for collection include: source code version, component set and grid resolution used for the simulation, environment variables, compiler name and machine specific information. User-modified files include configuration xml files, namelist input parameters for the model, and compiler flags used to create E3SM binaries (make file, modifications to source file, etc.). Finally, the script itself that was used to create, setup, and run a simulation is also captured. From this initial information, reproducibility will be achieved when a user is able to overwrite a default case study with the provenance information previously collected. In the second enrichment stage, we plan on extracting information from a subset of the ingested files (namelist input parameters and compiler flag

settings) to make each collection of E3SM provenance searchable and discoverable.

For scientific workflows such as those of the E3SM model, moving to an automated system of provenance capture not only means standardization of captured data but also allows for analysis of input parameters, allowing for fewer accidentally broken configurations and better utilization of machine allocations. The information captured by ProvEn includes git hash of the code used, the workflow component set and resolution, machine and compiler details, file name lists and job submission scripts. Scientists are encouraged to use a standardized “run_acme” script³ to capture this information, that is then ingested by the ProvEn provenance harvester. The run_acme.csh script is the script written by members of the E3SM team that handles all the setup and pre-run environment configuration needed to take place before the simulation starts. It creates the case studies directory structure, writes out all the parameters to file, and makes changes to the default namelist values automatically so users don’t have to manually edit the files. Before the run_acme script was created, each scientist used ad hoc scripts of their design or were performing the run setup manually so that most run configurations and parameters were not systematically stored and accessible. This led to a great deal of inconsistency and undesirable variability in runs from team to team. Consolidating all the setup into a single standardized script across teams for the entire project ensures that important configurations and inputs are captured, eases the burden on scientists, and improves reproducibility. The script captures many, but not all, the required inputs for complete bit-for-bit reproducibility, namely the namelists, xml parameters, machine parameters such as core count and node configuration. The ProvEn harvester captures code branch or compiler flags. The script has evolved to include new features, including a “git diff” comment section in the script itself capturing every code change that the scientist has made to their version of the model. This is required to capture small debugging changes that were committed but not pushed to the git repository and therefore not captured in the “git describe” feature. The ProvEn provenance harvester allows scientists to analyze ensemble runs to determine how changes in input parameters affect model output in a complex and nonlinear system. Provenance capture allows for reproducibility and a deeper analysis of the models performance characteristics across each code branch and compute facility.

Provenance harvesting and reproducibility was tested with an E3SM 4×4 case, a low resolution E3SM ne4 model. The simulation was run on edison.nersc.gov and the run_acme script was used to create, configure, and submit the case. The name of the component set used is FC5AV1C and the grid resolution is ne4_ne4. Once the simulation was completed, the ProvEn harvester was used to collect information described above and the configuration files. This information was then sent to the ProvEn semantic store. The collected information for that simulation was later retrieved from the ProvEn store and used to regenerate the

necessary configuration files for the simulation. The next step was to reproduce the simulation using the collected provenance information. The simulation is run on the same machine as previously and the standardized run_acme script is used to create a new case. The run_acme script used information from the retrieved provenance to point to the correct GitHub hashkey, component set, resolution, machine, and compiler. Now the configuration files are replaced with the regenerated files from ProvEn and the namelist files are replaced with the namelist files generated by ProvEn.

The next step is to evaluate reproducibility results and compare the original simulation to the reproduced one. This is done by comparing the output netcdf files from the runs using cprnc, an E3SM Fortran-90 tool built for this purpose.⁴ Cprnc provides a summary of the compared fields compared, details on fields that are found different, how many fields could not be analyzed, how many fields are found in one file and not in the other. In this test, the cprnc tool output confirmed that the corresponding netcdf files from the original and reproduced simulations are bit-for-bit identical. This test was done on a E3SM use case for prototyping. When tested with a higher resolution test case (ne30), our experiment showed that some variations in the directory structure needed to be additional captured by the scripts.

While bitwise reproducibility is a challenge and an elusive goal in parallel computing, these results show that it can be used by scientists to verify the correctness of their code on a limited set of test cases, rather than as a means to guarantee that the scientific research conducted with these codes is reproducible.

6.2. Performance reproducibility use case—LAMMPS

Large Scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) (Plimpton, 1995)⁵ is a widely used classical MDs simulation engine used for materials science studies that deploys MPI for parallel communication. We ran a series of experiments instrumenting this workflow and extracting metrics with Chimbuko to illustrate variations in performance in workflow total execution on an LCF system. Our LAMMPS workflow includes two components, the LAMMPS application without its analytics (C1—a producer) and a parallel writing component (stage-write or C2—a consumer) that uses an ADIOS API for optimal file writing performance. The workflow is configured with processes using dedicated cores for the LAMMPS calculations and the parallel writing component, and there is no resource contention between components. Each configuration was executed 10 times.

Figures 7 and 8 show the performance fluctuations for each configuration, with various numbers of processors per component, e.g. $128 + 32$, where 128 is the number of processors for C1 and 32 the number for C2. Figure 7 shows the average, minimum, and maximum workflow execution time of the 10 runs for each configuration. There

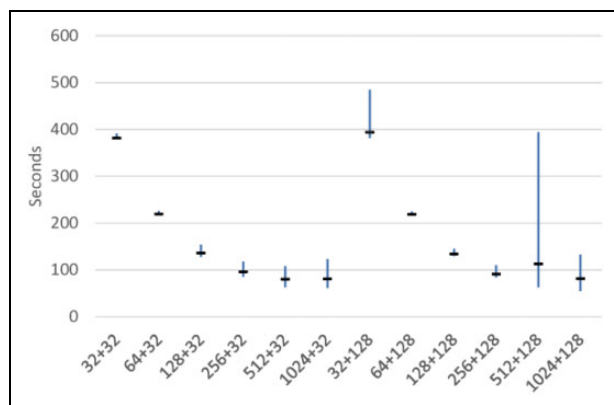


Figure 7. Average, maximum, and minimum workflow execution time of a simple LAMMPS workflow. LAMMPS: large-scale atomic/molecular massively parallel simulator.

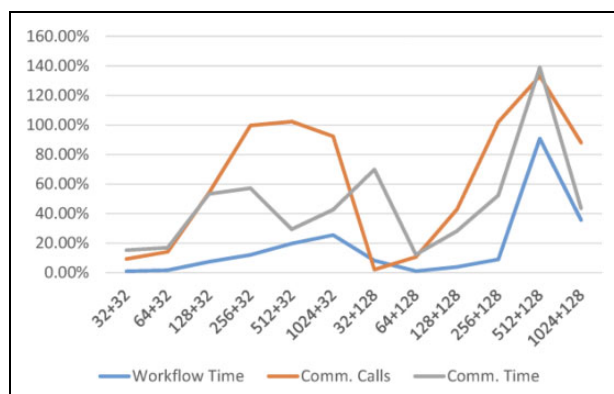


Figure 8. Standard deviation of workflow execution time and communication calls (MPI) for the simple LAMMPS workflow. LAMMPS: large-scale atomic/molecular massively parallel simulator.

are significant performance fluctuations for the configurations of $32 + 128$, $512 + 128$, and $1024 + 128$. The performance fluctuation decreases when the workflow uses relatively less processes. In Figure 7, the $32 + 128$ configuration shows significant performance lag. This is expected as the calculation component (C1) would typically use more resources than stage-write (C2). The $512 + 128$ outlier is unexplained by our experiment.

Figure 8 shows the standard deviation in percentages of (a) the total workflow execution time (blue curve), the numbers of workflow communication calls (orange curve), and the total workflow communication time (grey curve). Communication is measured as MPI calls. The number of communication calls has a lower standard deviation than the workflow time and the communication time for the $32 + 128$ configuration. This is caused by the runtime configuration where the number of producers is much lower than that of consumers, which causes delays for the consumer. Consumer wait time may also depend on the interconnect. Workflow time and communication calls are correlated (Pearson coefficient = 0.6945), and, except for this outlier, the total workflow execution time and the

numbers of communication calls share the same standard deviation trend. This supports the hypothesis that, in this case, communication overhead is a contributing factor to total execution time.

The performance fluctuations of workflows are dependent on many factors. As shown in Figures 7 and 8, using Chimbuko in the LAMMPS use case, we identified that communication time and MPI overhead are two strong factors contributing to workflow performance fluctuations. This communication overhead may be caused by both job configuration (e.g. number of nodes and topology) and neighboring concurrent large jobs. But establishing this as fact will require additional data not currently available to us. Although tools exist to obtain these data for a user runtime environment (Agrawal et al., 2014; Huang et al., 2016), such tools are conceived for facility operations and not for end-users and the data may be deemed too sensitive for open use. Community recommendations that organizational stakeholders and supercomputing centers take a more active role in supporting reproducible research and give end-users greater visibility into systems' real-time execution and network usage is pertinent here (James et al., 2014).

In addition, code instrumentation can introduce overhead dependent upon the type of instrumentation and the programming model used in a simulation. In typical cases, the overhead from TAU measurements is less than 1–2%, because either manual or automatic selective instrumentation helps eliminate the measurement of frequent, short-duration functions. A systematic and principled approach to provenance and performance metrics capture is needed to focus data collection on parts of interest, such as call stack execution, or communication methods.

7. Related work

A 1994 NIST report defines the repeatability of the results of measurements in terms of “the closeness of the agreement between the results of successive measurements of the same measure carried out under the same conditions of measurement.” Reproducibility is defined as such closeness “carried out under changed conditions of measurements” (Taylor and Kuyatt, 1994). Many studies have attempted to define reproducibility in computational science and list requirements. Contemporary studies focus on guidelines for publishing papers, appropriate use of statistical methods when reporting results, a more rigorous approach to the design of experiments, better documentation of the experimental process, and making data and source code available (Chirigati et al., 2013; Davison, 2012; Gil et al., 2016; Peng, 2011; Sandve et al., 2013). Bechhofer et al. (2010) and Goble et al. (2011) suggest that scientists communicate with Research Objects, rather than papers,—objects that bundle a workflow, its provenance traces, version history, author attribution, and more. Vetted benchmarks and trustworthy testbeds are designed for conducting reproducibility experiments in computational

science (Nussbaum, 2017a, 2017b; Vitek and Kalibera, 2012). Requirements for reproducibility are numerous and unclear and only started to be explored in details every step of a computational experiment (Carpen-Amarie et al., 2014). Recent recommendations include publishing source code, computational environments, and workflows in trusted repositories with persistent identifiers and links (Zhang et al., 2016a), as well as designing incentives to encourage reproducibility by journals and funding agencies (Stodden et al., 2016).

Performance reproducibility faces additional challenges. Large-scale computational environments are typically unique and rapidly changing. Multi-threading used in architectures can make program execution non-deterministic with performance variations due to the order in which memory is accessed on multi-core systems (Gramoli, 2016). Scientific replicability, which focuses on replicating the results of an experiment is seen as a more realistic goal by many while decried as unworthy by others (Drummond, 2009). In practice, it can require extensive modification of existing tools such as workflow management systems to run batches of simulations and capture the parallel execution environment (Hunold and Träff, 2013). The narrower goal of numerical reproducibility presents multiple challenges at the extreme scale, including those posed by fixed-point precision, floating-point operations because the round-off errors make floating point arithmetic non-associative (Collange et al., 2015; Demmel and Nguyen, 2013). Thus bitwise reproducibility would put severe constraints on the order of execution for instructions and data operations.

A DOE Office of Science workshop report on software productivity for extreme-scale science (Johansen et al., 2014) sought to direct computational practitioners to build a stronger foundation for more effectively exploiting and reusing resources for extreme-scale science. From a reproducibility perspective, the hope is that extreme-scale science is conducted with more consistency and based on well-understood community adopted methodologies. This is also one of the recommendations of the 2014 NSF XSEDE reproducibility workshop (James et al., 2014). A recent DOE Office of Science report on Extreme Heterogeneity defines “partial reproducibility,” where bitwise reproducibility can be achieved in debugging mode at much lower speed, but the constraint is relaxed in production runs (Vetter et al., 2018). In addition, the report stresses the importance of defining different levels of reproducibility appropriate for specific applications and study the trade-offs between reproducibility and performance. At extreme scale, reproducibility is entwined with performance portability, reliability, and resilience as Mean Time to Failure can potentially be measured in hours, and formal definitions of reproducibility should include performance factors.

Packaging libraries in easily deployable stacks and container technology such as Spack and Singularity help with the management of computing resources and improve

reproducibility (Gamblin et al., 2015). However, provenance still needs to be captured to enable analysis of provenance data. The problem of sharing and executing code among scientific users and the possibility of using virtual containers to overcome some computational reproducibility issues is discussed in (Boettiger, 2015). But virtualization schemes and the use of virtual containers at node level add latency to code execution and performance degradation at the interconnect (Zhang et al., 2016b). Other related tools include Sumatra, a python-based, reproducibility toolkit capturing workflow provenance and optimized for bioinformatics. The provenance information extracted is comprehensive, however, Sumatra does not capture performance metrics (Davison et al., 2014).

The Machine Learning (ML) community has been grappling with reproducibility for over 10 years with claims that publishing algorithms, implementations, and code as open source artifacts and publishing statistical thresholds for reproducing results should bring improvements. Recent efforts include making an entire research project open source with authorship determined by contribution,⁶ publishing benchmarks in data papers, logging decision points and training sets, encouraging the publication of make files, and the use of virtual machines (Langford and Larochelle, 2017).

8. Future directions for reproducible research

8.1. Adaptive workflows

Performance reproducibility is a difficult undertaking when the workflows are predetermined and well defined. If, however, the workflows themselves are non-deterministic, performance reproducibility and isolation becomes even harder. There are classes of workflows—which we refer to as adaptive workflows, which are defined by a task-graph that evolves based upon intermediate stages of computing. The task-graph of an adaptive workflow cannot be fully defined at the start of execution nor can the requirements of downstream computations be determined. The implications of adaptivity pervade multiple aspects of workflows—programming, deployment and resource management, and the predicted performance of workflows. Specifically, for adaptive workflows, discerning whether fluctuations are intrinsic to the workflow, or due to adaptive changes, or whether they arise due to “externalities” (e.g. performance fluctuations arising from dynamic variations in load and noise on multi-user platforms) of the workflow becomes a challenging and nontrivial undertaking.

We suggest that Workflow Building Blocks, based upon the end-to-end Radical Cybertools (Merzky et al., 2016) that decouple task dependencies in their order of execution from workload execution, can specify the information extraction needed for reproducibility. The building block approach introduced earlier addresses some of these

problems while proposing a viable alternative to the redesign and implementations of prototyped end-to-end systems. Specifically, a building block must be designed to be self-sufficient so that its design and performance does not depend on the details of other building blocks; interoperable so that it can be used in diverse system architectures without semantic modifications; composable so its interfaces enable communication and coordination with other building blocks; and extensible so that the building's block functionalities and entities can be extended to support new requirements or capabilities.

Whereas all four properties are important in the design and deployment of functional workflow systems, the property of self-sufficiency is critical to both isolate performance and support performance reproducibility. For example, the building block Radical-Pilot (Merzky et al., 2016) was used to the execution of the MD tasks in the experiments shown in Figure 2(a) and (b). Although the use of RADICAL-Pilot does not guarantee performance reproducibility in of itself, it enables the investigation and isolation of the source of fluctuations in performance.

8.2. In situ analytics

Storing trace files for complex simulations quickly becomes prohibitive due to size, so we designed a way to reduce this data, using prescriptive provenance (Pouchard et al., 2018). In situ analytics are being explored to reduce the large volume of data produced by trace files and summary information stored in performance profiles in Chimbuko. Not all provenance needs to be extracted online, nor all performance events need to be persisted. One method that we are exploring to reduce the size of performance data is to use change detection algorithms with in situ monitoring to store only events of interest for performance optimization of scientific workflows. We consider the extension of streaming manifold learning (Yoo et al., 2016) with anomaly score calculation (Huang et al., 2014) on the performance data. Current data reduction techniques within MD calculations rely on a mechanical sampling strategy, storing 1 out of 100 to 1000 time steps from a running simulation, thus possibly discarding meaningful structures from the output. Detecting and persisting "interesting" events online, for both scientific results and performance metrics while discarding the rest will reduce data. For Chimbuko, this means capturing decision points and thresholds as part of provenance when events are discarded.

Part of the research focus related to ProvEn has been exploring how traditional provenance analytics (searching and reasoning) can be improved through the use of ML techniques by replacing resource intensive on-demand queries with predictive modeling. In related work applicable to this problem, we have demonstrated training algorithms on empirical metrics from a real instrumented compute cluster, so that it is possible to detect abnormal

behavior and trends from streaming metrics and provenance (Singh et al., 2016).

9. Conclusion

The article has demonstrated that to enable scientific reproducibility, more than workflow-related information needs to be captured, namely additional provenance about the computer architecture and system software are critical. One of the main tasks in harvesting provenance for an application run is identifying the information to be captured for reproducibility. Our analysis has shown that extracted provenance data and performance metrics with state-of-the-art tools are useful to highlight fluctuations but may be too coarse to establish factors. Application communities will need to establish best practices, error bounds, and thresholds based on achievable levels of reproducibility for their applications at the extreme scale. Additional provenance information can then also enable new reproducibility use cases, such as performance reproducibility, increasingly important in many more operational computing settings such as real time experimental data analysis. Chimbuko and ProvEn are part of a first tool suite to address this exciting research area. While more work is needed, the presented use case studies have shown the utility of these tools.


Declaration of Conflicting Interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: The authors gratefully acknowledge the funding support from the US Department of Energy Office of Science/Office of Advanced Scientific Computing Research as part of "Integrated End-to-end Performance Prediction and Diagnosis (IPPD)". This research was partly supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration, "Co-Design Center for Online Data Analysis and Reduction." This manuscript has been authored by (a) employees of Brookhaven Science Associates, LLC under Contract No. DESC0012704, (b) employees of Pacific Northwest National Lab under Contract DEAC05-76RL01830. This work was also supported by the US Department of Energy Office of Science/Office of Biological and Environmental Research under Contract DE-AC52-07NA27344 at Lawrence Livermore National Laboratory.

ORCID iD

Line Pouchard  <https://orcid.org/0000-0002-2120-6521>

Notes

1. <https://github.com/CODARcode/Chimbuko>
2. <https://github.com/pnnl/ProvenanceEnvironment>
3. https://github.com/PeterCaldwell/run_acme
4. <http://www.cesm.ucar.edu/models/cesm1.0/clm/models/ln/clm/doc/UsersGuide/x8438.html>
5. <http://lammps.sandia.gov/index.html>
6. <http://ai-on.org>

References

- Agrawal K, Fahey MR, McLay R, et al. (2014) User environment tracking and problem detection with XALT. In: *Proceedings of the first international workshop on HPC user support tools*. pp. 32–40. IEEE Press. DOI: 10.1109/HUST.2014.6.
- Bechhofer S, De Roure D, Gamble M, et al. (2010) Research objects: towards exchange and reuse of digital knowledge. *Nature Proceedings*. DOI: 10.1038/npre.2010.4626.1.
- Boettiger C (2015) An introduction to Docker for reproducible research. *ACM SIGOPS Operating Systems Review* 49(1): 71–79.
- Carpen-Amarie A, Rougier A and Lübke FD (2014) Stepping stones to reproducible research: a study of current practices in parallel computing. In: *European conference on parallel processing EuroPar*. pp. 499–510. Springer. DOI: 10.1007/978-3-319-14325-5_43.
- Chirigati F, Shasha D and Freire J (2013) Packing experiments for sharing and publication. In: *Proceedings of the 2013 ACM SIGMOD international conference on management of data*, New York, USA, pp. 977–980. New York, USA: ACM. DOI: 10.1145/2463676.2465269.
- Cinquini L, Crichton D, Mattmann C, et al. (2014) The earth system grid federation: an open infrastructure for access to distributed geospatial data. *Future Generation Computer Systems* 36: 400–417.
- Collange S, Defour D, Graillat S, et al. (2015) Numerical reproducibility for the parallel reduction on multi- and many-core architectures. *Parallel Computing* 49: 83–97.
- Davison A (2012) Automated capture of experiment context for easier reproducibility in computational research. *Computing in Science & Engineering* 14(4): 48–56. DOI: 10.1109/MCSE.2012.41.
- Davison AP, Mattioni M, Samarkanov D, et al. (2014) Sumatra: a toolkit for reproducible research. In: *Implementing Reproducible Research*, pp. 57–79.
- Demmel J and Nguyen HD (2013) Numerical reproducibility and accuracy at exascale. In: *21st IEEE symposium on computer arithmetic*, pp. 235–237. IEEE. DOI: 10.1109/ARITH.2013.43.
- Drummond C (2009) Replicability is not reproducibility: nor is it good science. In: *Workshop on evaluation methods for machine learning at the international conference on machine learning (ICML)*. Montreal, Canada: International Machine Learning Society.
- Elsethagen T, Stephan E, Raju B, et al. (2016) Data provenance hybridization supporting extreme-scale scientific workflow applications. In: *New York scientific data summit (NYSDS)*, pp. 1–10. IEEE. DOI: 10.1109/NYSDS.2016.7747819.
- Gamblin T, LeGendre M, Collette MR, et al. (2015) The Spack package manager: bringing order to HPC software chaos. In: *2015 SC-international conference for high performance computing, networking, storage and analysis*, pp. 1–12. IEEE. DOI: 10.1145/2807591.2807623.
- Gil Y, David CH, Demir I, et al. (2016) Toward the geoscience paper of the future: best practices for documenting and sharing research from data to software to provenance. *Earth and Space Science* 3(10): 388–415.
- Goble C, De Roure D and Bechhofer S (2011) Accelerating scientists' knowledge turns. In: *International joint conference on knowledge discovery, knowledge engineering, and knowledge management*, pp. 3–25. Springer. DOI: 10.1007/978-3-642-37186-8_1.
- Gramoli V (2016) The information needed for reproducing shared memory experiments. In: *European conference on parallel processing*, pp. 596–608. Springer. DOI: 10.1007/978-3-319-58943-5_48.
- Heroux M, Barba LA, Parashar M, et al. (2018) *Toward a Compatible Reproducibility Taxonomy for Computational and Computing Sciences*. Sandia Technical Report: SAND2018-11186.
- Huang H, Qin H, Yoo S, et al. (2014) Physics-Based Anomaly Detection Defined on Manifold Space. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 9(2): 1–39. DOI: <https://doi.org/10.1145/2641574>.
- Huang R, Xu W and McLay R (2016) A web interface for XALT log data analysis. In: *Proceedings of the XSEDE16 conference on diversity, big data, and science at scale*, p. 31. ACM. DOI: 10.1145/2949550.2949560.
- Huck KA, Malony AD, Shende S, et al. (2006) TAUg: runtime global performance data access using MPI. In: *European parallel virtual machine/message passing interface users' group meeting*, pp. 313–321. Springer. DOI: 10.1007/11846802_44.
- Hunold S and Träff JL (2013) On the state and importance of reproducible experimental research in parallel computing. arXiv preprint arXiv:1308.3648.
- James D, Wilkins-Diehr N, Stodden V, et al. (2014) Standing together for reproducibility in large-scale computing: report on reproducibility @ XSEDE. arXiv preprint arXiv:1412.5557.
- Johansen H, McInnes L, Bernholdt D, et al. (2014) *Software Productivity for Extreme-Scale Science (2014)*. DOE ASCR Workshop Report, pp. 13–14.
- Johns M (2015) *Getting Started with Hazelcast*. Packt. ISBN: 1783554053.
- Kleese van Dam K, Stephan EG, Raju B, et al. (2015) Enabling structured exploration of workflow performance variability in extreme-scale environments. *8th Workshop in many-task computing on clouds, grids, and supercomputers (MTAGS) collocated with SC 2015*. Austin, TX: Pacific Northwest National Laboratory (PNNL), Richland, WA (US). DOI: 10.13140/RG.2.1.3311.9127.
- Koch D, Anantharaj V, Bader D, et al. (2016) Next-generation climate modeling science challenges for simulation, workflow and analysis systems. *AGU Fall Meeting Abstracts*. AGU abstract #IN13D-01.

- Langford J and Larochelle H. (2017) Reproducibility in Machine Learning. In: *34th International Conference on Machine Learning (ICML 2017)*. Reproducibility Workshop, Sydney, Australia, 6–11 August 2017.
- Liu Q, Logan J, Tian Y, et al. (2014) Hello adios: the challenges and lessons of developing leadership class i/o frameworks. *Concurrency and Computation: Practice and Experience* 26(7): 1453–1473.
- Merzky A, Santcroos M, Turilli M, et al. (2016) Executing dynamic and heterogeneous workloads on super computers. arXiv preprint:1512.08194v2.
- Moreau L and Groth P (2013) Provenance: an introduction to prov. *Synthesis Lectures on the Semantic Web: Theory and Technology* 3(4): 1–129.
- Nussbaum L (2017a) Testbeds support for reproducible research. In: *Proceedings of the reproducibility workshop*, pp. 24–26. ACM. DOI: 10.1145/3097766.3097773.
- Nussbaum L (2017b) Towards Trustworthy Testbeds thanks to Throughout Testing. In: *REPPAR-4th International Workshop on Reproducibility in Parallel Computing (IEEE IPDPSW)*, p. 9. DOI: 10.1109/IPDPSW.2017.101.
- Peng RD (2011) Reproducible research in computational science. *Science* 334(6060): 1226–1227.
- Pilgrim P (2015) *Digital Java EE 7 Web Application Development*. Packt. ISBN:1782176659.
- Plimpton S (1995) Fast parallel algorithms for short-range molecular dynamics. *Journal of Computational Physics* 117(1): 1–19.
- Pouchard L, Huck K, Matyasfalvi G, et al. (2018) Prescriptive provenance for streaming analysis of workflows at scale. In: *IEEE proceedings of the New York scientific data summit (NYSDS)*. Upton, NY: IEEE. DOI: 10.1109/NYSDS.2018.8538951.
- Pouchard L, Malik A, Van Dam H, et al. (2017) Capturing provenance as a diagnostic tool for workflow performance evaluation and optimization. In: *New York scientific data summit (NYSDS)*. New York, NY: IEEE. DOI: 10.1109/NYSDS.2017.8085043.
- Sandve GK, Nekrutenko A, Taylor J, et al. (2013) Ten simple rules for reproducible computational research. *PLoS Computational Biology* 9(10): e1003285.
- Singh A, Stephan E, Elsethagen T, et al. (2016) Leveraging large sensor streams for robust cloud control. In: *2016 IEEE international conference on Big Data (Big Data)*, pp. 2115–2120. IEEE. DOI: 10.1109/BigData.2016.7840839.
- Stodden V, McNutt M, Bailey DH, et al. (2016) Enhancing reproducibility for computational methods. *Science* 354(6317): 1240–1241.
- Taylor BN and Kuyatt CE (1994) *Guidelines for evaluating and expressing the uncertainty of NIST measurement results*. Gaithersburg, MD: US Department of Commerce, Technology Administration, National Institute of Standards and Technology. DOI: 10.1002/10.6028/NIST.tn.1297.
- Turilli M, Merzky A and Jha S (2016) Designing workflow systems using building blocks. arXiv preprint arXiv:1609.03484.
- Valiev M, Bylaska EJ, Govind N, et al. (2010) NWChem: a comprehensive and scalable open-source solution for large scale molecular simulations. *Computer Physics Communications* 181(9): 1477–1489.
- Vetter JS, Brightwell R, Gokhale M, et al. (2018) Extreme heterogeneity 2018-productive computational science in the Era of extreme heterogeneity. Report for DOE ASCR Workshop on Extreme Heterogeneity. USDOE Office of Science (SC)(United States). DOI: 10.2172/1473756.
- Vitek J and Kalibera T (2012) R3: repeatability, reproducibility and rigor. *ACM SIGPLAN Notices* 47(4a): 30–36.
- Yoo S, Huang H and Kasiviswanathan SP (2016) Streaming spectral clustering. In: *2016 IEEE 32nd international conference on data engineering (ICDE)*, pp. 637–648. IEEE. DOI: 10.1109/ICDE.2016.7498277.
- Zhang B, Pouchard LC, Smith PM, et al. (2016a) Data storage and sharing for the long tail of science. In: *IEEE proceedings scientific data summit (NYSDS)*, 2016 New York, pp. 1–9. DOI: 10.1109/NYSDS.2016.7747811.
- Zhang J, Lu X and Panda DK (2016b) High performance MPI library for container-based HPC cloud on InfiniBand clusters. In: *2016 45th international conference on parallel processing (ICPP)*, pp. 268–277. IEEE. DOI: 10.1109/ICPP.2016.38.
- Xie C, Xu W, Ha S, et al. (2018). Performance Visualization for TAU Instrumented Scientific Workflows. In: *Proceedings of the 13th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, Funchal, Madeira, Portugal, 27–29 January 2018. DOI: 10.5220/0006646803330340.

Author biographies

Line Pouchard is an information scientist who joined the Center for Data-Driven Discovery in the Computational Science Initiative at Brookhaven National Laboratory in January 2017. Prior positions include assistant professor at Purdue University, Staff Scientist at Oak Ridge National Laboratory, research associate professor and Advisory Board, School of Information Sciences, University of Tennessee, Knoxville, cofounder and lead of the DataONE Integration and Semantics Working Group. She led multi-disciplinary efforts improving scientific data discovery, data management and curation in scientific domains such as Earth and Environmental Sciences, Ecology, Materials Science, Extreme Scale Computing, Structural Engineering, and National Security. She has a PhD from the Graduate Center of the City University of New York and an MS from the University of Tennessee, Knoxville. Her research focuses on provenance, computational reproducibility, and text mining for big data. For more details see <https://linepouchard.github.io/profile/>

Sterling Baldwin has a degree in computer science and works at Lawrence Livermore National Lab for the E3SM modeling effort in the infrastructure and tools group.

Todd Elsethagen is currently working as a staff scientist at Pacific Northwest National Laboratory. His current

research interests include scientific data management, semantic web, and distributed data provenance capture and analysis.

Shantenu Jha is an associate professor of Computer Engineering at Rutgers University. He is also the Chair of the Center for Data Driven Discovery (C3D) as part of the Computational Science Initiative at Brookhaven National Laboratory. His research interests are at the intersection of high-performance distributed computing and computational science. He leads the RADICAL-Cybertools project which are a suite of middleware building blocks used to support large-scale science and engineering applications. He collaborates extensively with scientists from multiple domains—including but not limited to Molecular Sciences, Earth Sciences and High-Energy Physics. He is a recipient of the NSF CAREER Award (2013). More details can be found at <http://radical.rutgers.edu/shantenu>

Bibi Raju is a data scientist at Pacific Northwest National Laboratory (PNNL). She has been with PNNL since 2014. She received her MS in Computer Science from University of Texas at Arlington, TX. Her research interests include developing scientific provenance data management platform, knowledge management frameworks, reproducible research, real-time data streaming and data analytics, data hybridization and semantic web. Her interests also include developing visualization software for real-time simulation data and web development.

Eric Stephan received his Bachelor of Arts degree in Mathematics and Computer Science degree from Eastern Washington University in Cheney WA, USA, in 1986. He is a data scientist at the Pacific Northwest National Laboratory and has been active in the W3C data standards community. His research interests include scalable science and engineering data architectures, applied semantic web, experimental reproducibility and has been active in the data provenance community for over 12 years.

Li Tang is a staff scientist at Los Alamos National Laboratory and previously research associate of

Brookhaven National Laboratory. Before joining Brookhaven National Laboratory, he graduated with PhD from the Department of Computer Science and Engineering at University of Notre Dame. His research interests include hardware/software codesign, big data, performance/energy analysis and optimization, and heterogeneous computing.

Kerstin Kleese van Dam (recipient of the 2006 British Female Innovators and Inventors Silver Award) is director of the Computational Science Initiative at Brookhaven National Laboratory in the USA, leading BNL's computer science and mathematics R&D portfolio. Prior positions included associate Division Director, lead of the Scientific Data Management group, chief scientist and lead data services at the Pacific Northwest National Laboratory in the United States; director of Computing at the Bio-Medical Faculty at the University College London, UK; IT Program Manager and lead Scientific Data Management Group at the Science and Technology Facilities Council in the UK, HPC specialist at the German Climate Computing Center (DKRZ) and Software Developer at INPRO, a research institute of the German Automotive Industry. Kerstin has led a range of multi-disciplinary data management and analysis efforts in scientific domains such as Molecular Science (e-Minerals), Materials (e-Materials, Materials Grid, Chemical Imaging Initiative), Climate (DOE BER Accelerated Climate Modeling for Energy (ACME), DOE BER Climate Science for a Sustainable Energy Future (CSSEF), PNNL Platform for Regional Integrated Modeling and Analysis, NERC Data Services), Biology (DOE Bio Knowledgebase Prototype Project, Integrative Biology, BBSRC Archival Services), Chemical Imaging (Analysis in Motion, REXAN, ICAT), Power Grid (PNNL Future Power Grid Initiative) and High Energy Physics (BELLE II). She has over 100 peer-reviewed publications, book chapters and has edited a book on Data-Intensive Science. Her research is focused on data management and analysis in extreme scale environments. For more details see <http://www.linkedin.com/in/kerstinkleesevandam>