Reproducibility in Scientific Computing

Jonathan Klinginsmith

I. Introduction and Motivation

The Oxford English Dictionary defines the *scientific method* as "a method of procedure that has characterized natural science since the 17th century, consisting in systematic observation, measurement, and experiment, and the formulation, testing, and modification of hypotheses" [1].

Theory and *experimentation*, the first two pillars of the scientific method, have stood for centuries. Scientists have formulated theories and hypotheses and used experimentation to validate or refute theories. However in recent years, *computing* has widely been considered the third pillar of science [2]. Advances in sensors, imaging, and scientific instrumentation have led to generation of large amounts of scientific data. Data generation have become ubiquitous with science as well [3], to the point that some researchers consider data-intensive science to the fourth pillar [4].

Ivie and Thain define scientific computing "as computing applied to the physical sciences (biology, chemistry, physics, and so on) for the purposes of simulation or data analysis" [5]. Within the computational science research community, Stodden states "the digitization of science combined with the Internet create a new transparency in scientific knowledge, potentially moving scientific progress from building with black boxes, to one where the boxes themselves remain wholly transparent" [6].

As the scientific process further leverages both technology and data, the need to reproduce computational experiments has become imperative in the scientific discovery process. However, as a computational science researcher reading a scientific paper, it can be challenging to reproduce the authors' experiments. To fully reproduce the computational experiment, one must have the same versions of software installed and configured, have access to the original data, and leverage the same parameters used within the original experiment.

In many cases, having access to all these items is not possible [7]. Even if the original data are not available, it should be reasonable to expect experimental setup to be reproducible. Specifically, if the infrastructure setup and the software installation and configuration can be performed in a reproducible manner then scientists are much more enabled at replicating or extending the experiment in question.

Figure 1 models the progression of a computational science experiment. The three phases: *configuration*, *execution*, and *publication* represent logical constructs where experimental activities performed and can be replicated. During the *configuration* phase, software must be installed and configured and

when necessary infrastructure must be provisioned. This phase also includes any data preparation or downloads. Input parameters may be modified so that the experiment can be executed multiple times. Within the *execution* phase, the actual experiment is performed. Data and metrics are generated from experiment for use in the *publication* stage. In the *publication* stage, data tables, figures, and charts are produced for information sharing and presentation of experimental results.

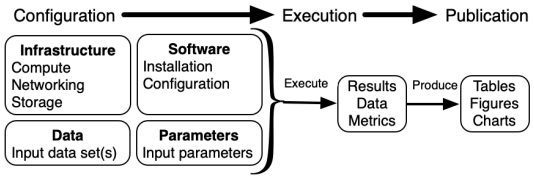

Fig. 1. Experimental progression

Many computational science disciplines are leveraging machine learning or artificial intelligence, in general, to make scientific inferences from trained datasets. Hutson [8] discussed the reproducibility crisis in artificial intelligence research. One of the most basic contributors to the crisis is researchers' lack of sharing and publishing software.

Heaven [9] provides reasons why AI is dealing with issues in reproducibility. He mentions one of most basic reasons for the crisis is the "lack of access to three things: code, data, and hardware." He continues that the divide between the "haves" and the "have-nots" of AI data and hardware is also contributing to the crisis.

In many cases as part of the experimental progression highlighted in in Figure 1 a scientific computing analysis requires multiple sequential or parallel steps. Scientific workflow software and platforms have emerged to enable researchers to define and execute these workflow steps. Reproducibility is also considered a requirement of scientific workflows [10].

II. Defining Reproducibility

The ability for researchers to reproduce the work of their peers has become imperative in the scientific discovery process. Researchers from a variety of scientific fields have called for the experimental data and code be made available such that published results can be conveniently reproduced [11].

Replicability and *reproducibility* have been defined in a number of ways throughout the research community.

Mitchell et al. define *replicability* as "the ability to run a code and produce exactly the same results as published" whereas *reproducibility* is referred to as "the ability to create a code that independently verifies the published results using the information provided" [12].

Madeyski and Kitchenham define *reproducible research* "as the extent to which the report of a specific scientific study can be reproduced (in effect, compiled) from the reported text, data and analysis procedures, and thus validated by other researchers" [13].

Drummond argues experimental replicability is not a substitution for true experimental reproducibility [14]. He provides an example of true experimental reproducibility where two researchers separately discover the speed of light is finite; coming to the

same experimental conclusion using two different experimental methods.

Hoefler and Belli define the notion experimental *interpretability*, which is considered a weaker concept than reproducibility. An "experiment is interpretable if it provides enough information to allow scientists to understand the experiment, draw own conclusions, assess their certainty, and possibly generalize results" [15].

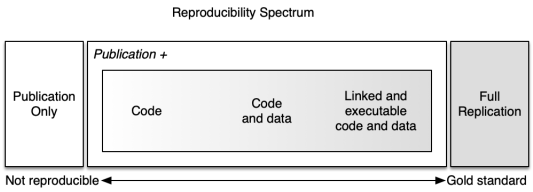

Fig. 2. The spectrum of reproducibility from [16]

There is a spectrum when it comes to considering replicability, and this spectrum applies to reproducibility as well. Figure 2 covers the spectrum. *Publication only* is at one end, which indicates a lack of access to the experimental data and software. An experiment is considered not reproducible at the *publication only* end of the spectrum. *Full replication* is at the other end of the spectrum, which indicates software and data are accessible and the experiment is fully replicable. Full replication is the standard to strive for.

The bare minimum researchers should provide are the software and data used in the experiments [16]; however, these two items together do not truly provide reproducibility. To fully reproduce a computational science experiment, several items from the original experiment must be considered. Initially, one must have the software configured in the same manner as the original experiment. Access to the original experimental data may not always be possible, but its usage also aides in reproducibility. It is further necessary to provide information on how to execute the experiment, such as how to run the software and what the appropriate parameters are.

Within the reproducibility spectrum, accessibility to data plays an important role in striving for full reproducibility. Wilkinson et al. introduces the FAIR principles for scientific data [17]. FAIR, which stands for *Findable*, *Accessible*, *Interoperable*, and *Reusable*, are guiding principles and should be considered when enabling the reuse of scientific data.

III. Contributing Factors to Reproducibility

This section highlights technologies, research areas, and other topics that contribute to reproducibility.

A. Programming Languages and Software

Two prominent programming languages in scientific computing, Python and R, have introduced tools and systems to aid in reproducibility.

For the R programming language, Madeyski and Kitchenham discuss how the use of the integrated development environment RStudio and the `knitr` R package are tools that aid in reproducibile research [13]. Reproducibility and portability in the R programming language are two of the key motivations for Packrat [18], a system for package dependency management. Packrat keeps track of each installed package and its version.

Python's package installer tool, pip [19], provides a requirements file convention that allows for package dependencies and versions to be specified. The `pip freeze` command acts in the same manner as Packrat with R by producing a listing of packages and versions, acting to snapshot the dependencies of the environment. The `pip freeze` export can be used with the `pip install` command to install and configure a new Python environment with the same package versions. It is standard practice to write the output of the `pip freeze` command to a requirements file named *requirements.txt* [20].

Software module systems such as environment modules[21] and lmod[22] [23] were created by administrators of high performance computing clusters to manage versions of computational software, programming language compilers, numerical and other system libraries. Scientists using these module systems can load the necessary artifacts, including specific versions to create an environment to run a computational set of tasks.

Computational reproducibility at the Texas Advanced Computing Center (TACC) was discussed in the following podcast [24]. The use of lmod[22] was discussed within the podcast as a means to enable reproducibility on their supercomputers.

B. Virtualization

Virtualization technology has opened the door to many advances in computing. A virtual machine (VM) [25] is a running instance of a computer where resources such as memory and central processing units (CPUs) are allocated through virtualization software. Virtual machines are instantiated from a virtual machine *image*. It is possible to export the state of a running virtual machine into an image.

A *virtual appliance* is an exported virtual machine image with the pre-installed and configured software included. Several works [26, 27] have outlined how virtual appliances can provide a reproducible environment for researchers. Dudley and Butte propose reproducibility as a two step process [26]. First, by storing scientific datasets in the cloud more researchers have access to the data. Second, the authors introduce a concept called Whole System Snapshot Exchange (WSSE) where the entire computer system, including the operating system, is copied so that other researchers can fully replicate the *in silico* experiment. Howe shares this same sentiment [27]. The author states that by providing a pre-installed, pre-configured virtual machine, the original "laboratory" is provided intact for future researchers.

C. Cloud Computing

Cloud computing has emerged in recent years because of advances in virtualization software. Companies such as Amazon, Google, and Microsoft provide services to customers for use of the virtual resources owned by them. The definition of cloud computing has taken many forms in the academic community and industry. For this work, we use the definitions and terms discussed in [28].

Cloud computing refers to the software, platforms, and infrastructure services provided over the Internet as well as the data centers offering these ser-

vices [29].. The hardware and virtualization software running on top of this hardware is termed a *cloud*. Providers of cloud computing services offer application programming interfaces (APIs) as means with which customers subscribe and interact. The lowest level of service available with a cloud is termed *Infrastructure as a Service* (IaaS). At this level, users provision virtual infrastructure resources such as a virtual machine or a virtual block storage device through an application programming interface (API). A category of software called *infrastructure as code* (IaC) software was created to interact with the cloud APIs. AWS created its own IaC software called CloudFormation [30]. Similarly, Microsoft Azure created its own IaC software called ARM [31]. Terraform is an IaC software tool created independent of clouds, but provide plugins to interact with a particular IaaS cloud [32]. All of these IaC software tools provide the ability to create software-defined reproducible cloud infrastructure.

The use of cloud computing has been discussed by researchers as a means to aid in the reproducibility of *in silico* experiments [26, 27]. Howe demonstrates how virtual appliances, which are virtual machines with pre-installed and pre-configured software, can be used for reproducible research [27]. The concept of a software configuration snapshot can be applied across a variety of technology stacks. Just as a virtual machine appliance preserves the pre-configured software, containers can also be considered a snapshot of a software environment. Containers are discussed in detail in the next section.

D. Containers

Advances in the Linux operating system kernel provided the underpinnings for containers. A container is a packaged set of software and its corresponding dependencies. Unlike a virtual machine, a container does not need virtualization software to run, but rather run directly on the operating system of the host machine.

Docker [33] is the most populate container ecosystem. Singularity [34], introduced by the high performance computing community as a container technology, can be executed without administrator access and without a daemon process. Both Docker and Singularity have commands that can be executed to create the container. With Docker, the convention is to create a Dockerfile to execute. Similarly, Singularity has a concept of a definition file which contains the set of instructions to execute to build the container. Singularity and Docker were compared in the context of HPC applications in [35]. The authors performed several benchmark comparisons.

The Open Container Initiative (OCI) [36] was created to align container formats and container runtime environments to an open standard. Initiatives such as the OCI allow for further community conformity and aid in reproducibility.

Using Docker for reproducibility in software engineering research is discussed in [33]. Boettiger discussed how containers help overcome four reproducibility challenges associated with scientific computing: *dependency hell*, *imprecise documentation*, *code rot*, and *barriers to adoption and reuse*

in existing solutions [37].

A common process when building the container is to version it. Both Docker and Singularity provide web-accessible hosting repositories so that the versioned container artifact can be downloaded for future usage. In the next section, artifacts and artifact repositories are discussed in further detail.

E. Artifacts

An *artifact* is an individual software or data product created as part of a process. For software, an artifact can be a package for a programming language, a package for an operating system, a software container, among other possible items. The artifact may be created as part of compiling and packaging software into an individual unit. For data, the artifact

The following properties are important when defining an artifact. First, the artifact must be versioned or it must contain a unique identifier. By versioning the artifact, it is possible to capture a point-in-time snapshot of it. Second, the artifact must be accessible. Typically by hosting the artifact from a downloadable location, it can be accessed. A hosting location for a set of artifacts, an *artifact repository*, allows artifacts to uploaded, stored, and distributed. Artifact repositories may also contain capabilities for indexing artifacts, searching for artifacts, among other advanced and specific functionality.

Programming languages such as Python and R, host their software packages in public repositories. For both languages, the package contains a specific software version. The R programming language has two main package repositories. CRAN [38] is the primary and most widely used R package repository. Bioconductor [39] is another R package repository. Bioconductor hosts bioinformatics packages and reference data. Python also has a public repository named PyPI, short for Python Package Index[40].

The Linux operating system has several different distributions. Each distribution has created a software package format for installation of operating system libraries. The distribution providers host packages in web-accessible repositories and provide software tools for installing, updating, and removing the packages.

Docker Hub [41] and Singularity Hub[42] are the container repositories for Docker and Singularity, respectively. Each host version container images that are available for software developers to use or from which to build a new container.

Tools such as Artifactory [43] provide a universal repository management platform to either proxy the public repositories or create private repositories. Public clouds are also hosting cloud-native artifact repositories. Amazon Web Services' hosted artifact repository is called *AWS CodeArtifact* [44]. Microsoft Azure's hosted artifact repository is called *Azure Artifacts* [45].

Software artifacts enable reproducibility because they provide a reusable, versioned, and downloadable software product that is available for future usage. The artifact provides a point-in-time snapshot of the software packaged within it.

Data can also be considered an artifact. Providing unique URLs, versions, or tags allows researchers

the ability to access a uniquely identifiable version of the data set. The concept of keeping a versioned copy of data for a machine learning system is discussed in [46].

The FAIR guiding principles discuss the use of artifact identifiers for data. For data to be findable, it must have a "globally unique and persistent identifier" [17]. Moreover, for data to be accessible, they must be "retrievable by their identifier using a standardized communications protocol" [17]. In general, the FAIR principles "act as a guide to data publishers and stewards to assist them in evaluating whether their particular implementation choices are rendering their digital research artefacts Findable, Accessible, Interoperable, and Reusable" [17].

F. DevOps

The advent of cloud computing has enabled software developers to provision virtual infrastructure on demand, thus eliminating the need to work with an operations team to procure or administer the infrastructure. The portmanteau *DevOps* was created from the words *developer* and *operations*, signifying the unification of software development responsibilities with operational responsibilities. A key component of DevOps is automation and software tools, in particular, have enabled automation spanning software operations and development.

Configuration management tools such as Ansible [47], CFEngine [48], Chef [49], Puppet [50], and SaltStack [51] aid in the automation of installation and configuration of software and were created as a result of cloud computing. These tools enable software developers to recreate software environments because the installation and configuration steps are maintained software automation versus the steps being outlined in documentation. The use of these tools enables researchers to reproduce a software environment setup for a particular experiment.

Git [52] is a popular distributed version control system, which enables software developers to version their software. The versioning of software is a fundamental underpinning to reproducibility because it provides a specific reference point for current or future usage. Github [53] is a Git platform that allows software developers to publicly or privately host their software project repositories. Automated deployments of infrastructure and versioned software configurations are now performed via continuous integration (CI) and continuous delivery (CD) [54] pipelines.

G. Computational Notebooks

The notion of documenting ones scientific work is not new. Documentation can take many forms. One example of documentation is the peer-reviewed process of scientific publications. Researchers publish the work performed along with the results obtained. Another form of documentation, is the written step-by-step process a researcher has gone through to perform an experiment. The use of a scientific notebook to document the step-by-step experimental process is a standard practice.

Schnell discusses ten best practise rules for a computational biologist's lab notebook. Rule six: "Keep a Record of How Every Result Was Pro-

duced" is directly related to repoducibility. Rule seven: "Use Version Control for Models, Algorithms, and Computer Code" discussed how good software engineering practices can be referenced within the notebook [55].

The IPython [56] community created a set of tools for interactive usage of the Python programming language. The Jupyter [57] notebook is a popular computational notebook created from the IPython community. Although originally created for the Python language, the Jupyter notebook is now used by many programming languages within the data science and scientific computing space, such as Julia and R.

Binder [58] is a project that builds a Github repository into a Jupyter computational notebook. It supports notebooks written in Julia, Python, or R. It is possible for a person to execute code when exporting the computational notebooks or alternatively to execute code when re-instantiating the notebook. This flexibility allows for custom scripts or data to be included in the Docker image or alternatively for the custom script to be executed or data to be downloaded at run-time.

H. Clusters

A computing cluster is a set of computers, attached storage, and networking that operates as a larger logical entity to solve highly complex scientific computing problems. Clusters can be comprised of physical hardware, but the properties of scalability and elasticity have gained much of cloud computing's attention, which has led to work in building virtual clusters. A virtual cluster [59] is a set of virtual machines and any corresponding storage, which operate as a whole to create the presence of a single computational entity. The ability to quickly provision a virtual cluster within cloud computing infrastructure is useful in many reproducibility scenarios within computational science experiments. For example, a researcher can build the necessary environment closer to the data, especially as more public datasets are being stored in the cloud [60]. Both [61, 62] discuss the how as the size of data sets grow, and if the construction of a virtual cluster can be performed in a straightforward manner, then it is possible to move the computing infrastructure to the data. This is in contrast to moving the data to the location of a dedicated high performance computing (HPC) cluster, such as a research university's supercomputer.

Other researchers have worked on bridging the gap between dedicated physical clusters and the elasticity of the cloud [63].

I. Comparisons

- **Programming languages** - Languages such as Python and R both provide mechanisms to package software as well as replicate the installed environment. Con: The language and packages must be installed and configured. Con: A researcher must install the base language along with all of the package dependencies.
- **Virtual Appliances** - A virtual appliance prepackages an entire software environment

Category	Pr.	Co.	So.	Di.	Do.	Ve.	Pa.	Lg.
Programming Languages							Y	
Virtual Appliances			Y					
Containers			Y					
Source Code Management Software						Y		
Configuration Management Software		Y						
Infrastructure as Code Software	Y							
Artifact Repositories				Y				
Computational Notebooks					Y			
Clusters								Y

TABLE I
PR. = PROVISIONS INFRASTRUCTURE; CO. = CONFIGURES INSTANCE; SO. = SOFTWARE PRE-INSTALLED; DI. = DISTRIBUTES PACKAGED SOFTWARE; DO. = DOCUMENTATION AND CODE; VE. = VERSIONS SOFTWARE; PA. = SOFTWARE PACKAGING; LG. = LARGE-SCALE DISTRIBUTED USE-CASES

and operating system. Con: Data are not typically stored with the virtual machine image, but can be done at smaller scales. Con: You need virtualization software or a cloud to run a virtual machine. Con: It is also more difficult to make changes, save, and publish a virtual appliance for others to use.

- **Containers** - Containers also have software preinstalled and configured like a virtual appliance. It is easier to build a container provide versioned, packaged components. Con: Need a runtime environment such as Docker or Singularity or a container orchestration platform such as Kubernetes to run. Con: Data are not typically packaged with the container, but can be done at smaller scales.
- **Source code management software** - In a traditional sense, source code management software is used to version software; platforms like Github host source code and are adding more functionality to perform software compilation and builds through continuous integration/continuous deployment (CI/CD). Con: The software must be deployed to a target system (e.g., virtual appliance, container, or cluster).
- **Configuration management software** - can install and configure software, including base programming language and language packages. Con: does not spin up infrastructure, if using IaaS cloud APIs.
- **Infastructure as code software** - used to interact (e.g., instantiate or terminate) with cloud APIs, in particular Infrastructure as a Service cloud APIs. Con: Either configuration management still necessary to install and configure within a running virtual server or either a virtual appliance or container is needed with prepackaged software.
- **Artifact Repositories** - Artifact repositories host and distribute versioned, packaged components such as programming language packages, container images. Con: An installation environment is still needed to run the software.
- **Computational Notebooks** - Notebooks are able to mix code and documentation; can execute code directly from the notebook. Con:

there are some limitations to code that can be executed within the notebook.

- **Clusters** - Clusters are more complex to install and configure because of being a distributed environment, but able to reproduce larger scale scenarios. Con: Still need software installed and configured.

IV. Benchmarks, Experimentation, and Testing

This section discusses the role and ties of reproducibility in benchmarks, experimentation, and testing.

A. Benchmarks

A computational benchmark brings conformity to a problem, an experiment, or an analyses. It serves as a means for performing evaluations, comparisons, and measurements. The concept of benchmarking is not new. Two such benchmarks, TPC and SPEC, have have existed for many years. The Transaction Processing Performance Council (TPC™) "is a non-profit corporation focused on developing data-centric benchmark standards and disseminating objective, verifiable performance data to the industry." [64] The Standard Performance Evaluation Corporation (SPEC) "is a non-profit corporation formed to establish, maintain and endorse standardized benchmarks and tools to evaluate performance and energy efficiency for the newest generation of computing systems." [65].

Benchmarks are actively being developed in the areas of artificial intelligence and machine learning. In [66], the authors introduce Deep500, a *meta-framework* that is able to support various deep learning benchmarks. Deep500 was created with five pillars in mind: customizability, metrics, performance, validation, and reproducibility.

MLPerf[67] is a more recent benchmarking addition to the machine learning community. Both the SPEC benchmark for general purpose computing as well as the TPC benchmark for database systems were motivations for the MLPerf benchmark [67].

Bourrasset discussed how the concept of *reproducibility* as an important characteristic of good benchmarks and metrics [68]. Because of the necessity to execute a benchmark numerous times when performing comparisons, it is crucial the benchmark is reproducible. However, benchmarking measurements can be difficult to reproduce, especially in the area of artificial intelligence. The The time-to-accuracy (TTA) measurement introduced in DAWNBench [69] is used as an evaluation criteria. This measurement makes it easier to compare different machine learning approaches. It also eliminates any replication issues.

B. Experimentation and Reproducibility

Four properties of good experimentation are outlined in [70]: 1) reproducibility, 2) extensibility, 3) revisability, and 4) applicability. An experiment exhibits the property of *reproducibility* if another researcher is able to perform the experiment and produce the same result. The properties of *extensibility* and *revisability* both dictate that changes to conditions and modifications must be possible for

the experiment. Those changes may be necessary for future expandability or if a modification is needed for corrections to the original experiment. Lastly, experimental parameters must be of realistic in nature and resembling real-world conditions while allowing for changes to thus promote *applicability*.

Desprez et al. discussed experimental complexity and scalability [71]. In order to answer the most difficult of scientific questions, an experiment must meet the complexity and scalability demands of the problem. Therefore, advancements in managing ever more complicated experiments will lead to improvements in reproducibility and experimental integrity.

C. Testbeds

Testbeds have been created to allow researchers across a variety of scientific domains the ability to perform experiments. Several testbeds have been created across academic institutions to support experimentation. Jetstream [72] is a cloud-based environment for researchers to leverage scientific computing domains. Chameleon Cloud [73] is a testbed used cloud computing research and experimentation. The FutureGrid [74] testbed was created for researchers to do complex computer science experimentation in areas such as cloud computing, grid computing, and and high performance computing. Overall, testbeds play an important role in reproducible experimentation [75].

V. Limitations and Challenges to Reproducibility

This section discusses limitations and challenges to reproducibility.

A. Limitations

There are many technical limitations to reproducibility. The use of stochastic methods, such as the use of genetic algorithms or Monte Carlo simulations, can prevent an experiment from being fully reproducible [76]. The randomness embedded within these methods make reproducibility difficult. The finite nature of floating point precision also limits numerical simulation reproducibility. It is entirely possible to produce a different results from a commutative mathematical operation, such as multiplication, when modifying the order of the floating point factors. Hoefler and Belli discuss how performance may not be reproducible in the domain of parallel computing, but *interpretability* can still foster advancements in scientific knowledge [15].

B. Challenges

There are many challenges to overcome in striving for reproducibility. One of the single greatest challenges is overcoming the cultural barrier in sharing one's code and work [37]. The academic community is not incentivized to publish code and data in a manner to make it reproducible for a future researcher. Figure 2 highlights the reproduciblity spectrum, and as one moves from *publication only* across the spectrum to *full replication*, the more time and effort a researcher has to contribute to the reproducibility efforts.

Access to very restrictive or expensive infrastructure as well as proprietary data is also a barrier to reproducibility. Time allocations for computing and experimentation on world-class supercomputers and instruments is very limited and very costly. Thus it can be difficult to reproduce experiments by another researcher on the equipment. The lack of access to software, data, and hardware is preventing reproducibility in AI [9].

VI. Conclusion

The ability for researchers to reproduce the work of their peers has become imperative in the scientific discovery process. As computing has become a fundamental part of the scientific process, so to has the need to reproduce scientific computing experiments. There are many factors aiding in the ability to strive for reproducible computational experiments. Applying best practices in software engineering, such as the use of automation, versioning software, among others enable researchers to build on the work of others. Advancements in areas of cloud computing, virtualization, and containers has also enabled researchers access to software-defined infrastructure and preconfigured software environments, thus enabling reproducibility.

Unfortunately, there are challenges to reproducibility as well, starting the foundational barriers of getting researchers to recognize the benefits to setting up a computational experiment for the future. Additionally, data introduces complexity to experiments. Having access to the original experimental data can be challenging and thus limit reproducibility.

Overall, progress has been made in recognizing and tackling the issues of reproducibility in the area of scientific computing. It is an area of continued progress and active research.

References

[1] *Oxford English Dictionary*. Oxford University Press, 2019.

[2] Gordon Bell, Tony Hey, and Alex Szalay. “Beyond the Data Deluge”. In: *Science* 323.5919 (2009), pp. 1297–1298.

[3] “Data, data everywhere: A special report on managing information”. In: *The Economist* (2010), pp. 1–14.

[4] Tony Hey, Stewart Tansley, and Kristin Tolle. “Jim Gray on eScience: A Transformed Scientific Method”. In: *Microsoft Research* (2009).

[5] Peter Ivie and Douglas Thain. “Reproducibility in Scientific Computing”. In: *ACM Comput. Surv.* 51.3 (July 2018), 63:1–63:36.

[6] Victoria Stodden. *The Scientific Method in Practice: Reproducibility in the Computational Sciences*. Tech. rep. 4773-10. MIT Sloan School of Management, 2010.

[7] Science Staff. “Challenges and Opportunities”. In: *Science* 331.6018 (2011), pp. 692–693.

[8] Matthew Hutson. “Artificial intelligence faces reproducibility crisis”. In: *Science* 359.6377 (2018), pp. 725–726. eprint: https://science.

sciencemag.org/content/359/6377/725.full.pdf.

[9] Will Douglas Heaven. *AI is wrestling with a replication crisis*. https://www.technologyreview.com/2020/11/12/1011944/artificial-intelligence-replication-crisis-science-big-tech-google-deepmind-facebook-openai/. 2020.

[10] Yolanda Gil et al. "Examining the Challenges of Scientific Workflows". In: *Computer* 40.12 (2007), pp. 24–32.

[11] Victoria Stodden, Ian Mitchell, and Randall LeVeque. "Reproducible Research for Scientific Computing: Tools and Strategies for Changing the Culture". In: *Computing in Science and Engineering* 14.4 (2012), pp. 13–17.

[12] Ian Mitchell, Randall LeVeque, and Victoria Stodden. "Reproducible Research for Scientific Computing: Tools and Strategies for Changing the Culture". In: *Computing in Science and Engineering* 14.4 (July 2012), 13–17.

[13] Lecha Madeyski and Barbara Kitchenham. "Would wider adoption of reproducible research be beneficial for empirical software engineering research?" In: *Journal of Intelligent & Fuzzy Systems* 32.2 (2017), pp. 1509–1521.

[14] Chris Drummond. "Replicability is not Reproducibility: Nor is it Good Science". In: *Proc. of the 26th Intl. Conf. of Machine Learning, Evaluation Methods for Machine Learning Workshop*. 2009.

[15] Torsten Hoefler and Roberto Belli. "Scientific Benchmarking of Parallel Computing Systems: Twelve Ways to Tell the Masses When Reporting Performance Results". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '15. Association for Computing Machinery, 2015. ISBN: 9781450337236.

[16] Roger D. Peng. "Reproducible Research in Computational Science". In: *Science* 334.6060 (2011), pp. 1226–1227.

[17] Mark D. Wilkinson et al. "The FAIR Guiding Principles for scientific data management and stewardship". In: *Scientific Data* 3.160018 (2016).

[18] *Packrat*. https://rstudio.github.io/packrat/. 2019.

[19] *pip - The Python Package Installer*. https://pip.pypa.io/en/stable/. 2019.

[20] *The Python Tutorial: Virtual Environments and Packages*. https://docs.python.org/3/tutorial/venv.html. 2020.

[21] John L. Furlani. "Modules: Providing a Flexible User Environment". In: *Proceedings of the Fifth Large Installation Systems Administration Conference (LISA V)*. 1991, pp. 141–152.

[22] M. Geimer, K. Hoste, and R. McLay. “Modern Scientific Software Management Using EasyBuild and Lmod”. In: *2014 First International Workshop on HPC User Support Tools*. 2014, pp. 41–51.

[23] *Lmod: A New Environment Module System*. 2020.

[24] *Science On Repeat - Computational Reproducibility*. https://soundcloud.com/usetacc/science-on-repeat-computational-reproducibility. 2019.

[25] Robert Goldberg. “Survey of Virtual Machine Research”. In: *IEEE Computer*. 1974, pp. 34–45.

[26] Joel Dudley and Atul Butte. “In silico Research in the Era of Cloud Computing”. In: *Nature Biotechnology* (2010), pp. 1181–1185.

[27] Bill Howe. “Virtual Appliances, Cloud Computing, and Reproducible Research”. In: *Computing in Science and Engineering* 14 (2012), pp. 36–41.

[28] Michael Armbrust et al. *Above the Clouds: A Berkeley View of Cloud Computing*. Tech. rep. UCB/EECS-2009-28. University of California at Berkeley, 2009.

[29] Michael Armbrust et al. “A View of Cloud Computing”. In: *Commun. of the ACM* 53.4 (2010), pp. 50–58.

[30] *AWS CloudFormation - Infrastructure as Code & AWS Resource Provisioning*. https://aws.amazon.com/cloudformation/. 2021.

[31] *Azure Resource Manager documentation*. https://docs.microsoft.com/en-us/azure/azure-resource-manager/. 2021.

[32] *Azure Resource Manager documentation*. https://www.terraform.io/. 2021.

[33] J. Cito and H. C. Gall. “Using Docker Containers to Improve Reproducibility in Software Engineering Research”. In: *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*. 2016, pp. 906–907.

[34] *Singularity info*. https://www.sylabs.io/singularity/.

[35] Pankaj Saha et al. “Evaluation of Docker Containers for Scientific Workloads in the Cloud”. In: *Proceedings of the Practice and Experience on Advanced Research Computing*. PEARC ’18. Pittsburgh, PA, USA: Association for Computing Machinery, 2018.

[36] *Open Containers Initiative: Home*. https://www.opencontainers.org/. 2020.

[37] Carl Boettiger. “An Introduction to Docker for Reproducible Research”. In: 49.1 (Jan. 2015), 71–79.

[38] *The Comprehensive R Archive Network*. https://cran.r-project.org/.

[39] *Bioconductor*. https://www.bioconductor.org/.

[40] *PyPI – the Python Package Index*. https://pypi.org/. 2019.

[41] *Docker Hub*. https://hub.docker.com/.

[42] *Singularity Container Registry*. https://singularity-hub.org/.

[43] *Artifactory - Universal Artifact Repository Manager*. https://jfrog.com/artifactory/. 2019.

[44] *AWS CodeArtifact - Amazon Web Services*. https://aws.amazon.com/codeartifact/.

[45] *AWS CodeArtifact - Amazon Web Services*. https://azure.microsoft.com/en-us/services/devops/artifacts/.

[46] David Sculley et al. "Hidden technical debt in machine learning systems". In: *Advances in neural information processing systems*. 2015, pp. 2503–2511.

[47] Daniel Hall. *Ansible configuration management*. Packt Publishing Ltd, 2013.

[48] *Configuration management software | Open source configuration management - CFEngine - Distributed Configuration Management*. http://cfengine.com.

[49] *Chef: Enabling the Coded Enterprise through Infrastructure, Security and Application Automation*. https://www.chef.io/.

[50] *Puppet Labs: IT Automation Software for System Administrators*. http://www.puppetlabs.com/.

[51] *Home — SaltStack*. https://www.saltstack.com/.

[52] https://git-scm.com/. 2020.

[53] https://github.com/. 2020.

[54] Isaac Sacolick. *What is CI/CD? Continuous integration and continuous delivery explained*.

[55] Santiago Schnell. "Ten Simple Rules for a Computational Biologist's Laboratory Notebook". In: *PLoS computational biology* 11.9 (Sept. 2015).

[56] https://ipython.org/. 2020.

[57] https://jupyter.org/. 2020.

[58] https://mybinder.org/. 2020.

[59] Ian Foster et al. "Virtual Clusters for Grid Communities". In: *Proc. of the 6h IEEE Int. Symp. on Cluster Computing and the Grid*. 2006, pp. 513–520.

[60] NIH. *1000 Genomes Project data available on Amazon Cloud*. Press Release. http://www.nih.gov/news/health/mar2012/nhgri-29.htm. 2012.

[61] Simone Leo et al. "Using Virtual Clusters to Decouple Computation and Data Management in High Throughput Analysis Applications". In: *Proc. of the Euromicro Conference on Parallel, Distributed, and Network-Based Processing*. 2010, pp. 411–415.

[62] Paolo Anedda et al. "Suspending, migrating and resuming HPC virtual clusters". In: *Future Generation Computer Systems* 26.8 (2010), pp. 1063 –1072.

[63] H. Kim and M. Parashar. "CometCloud: an autonomic Cloud engine". In: *Cloud Computing: Principles and Paradigms*. Wiley, 2011. Chap. 10.

[64] http://www.tpc.org/. 2019.

[65] https://www.spec.org/. 2019.

[66] Tal Ben-Nun et al. “A Modular Benchmarking Infrastructure for High-Performance and Reproducible Deep Learning”. In: *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 2019, pp. 66–77.

[67] *MLPerf*. https://www.mlperf.org/. 2019.

[68] Cedric Bourrasset et al. “Requirements for an Enterprise AI Benchmark”. In: *Performance Evaluation and Benchmarking for the Era of Artificial Intelligence*. Ed. by Raghunath Nambiar and Meikel Poess. Cham: Springer International Publishing, 2019, pp. 71–81. ISBN: 978-3-030-11404-6.

[69] Cody Coleman et al. “Analysis of DAWNBench, a Time-to-Accuracy Machine Learning Performance Benchmark”. In: *SIGOPS Oper. Syst. Rev.* 53.1 (July 2019), 14–25.

[70] Team. “Algorithms for the Grid: INRIA Research proposal”. http://www.loria.fr/equipes/algorille/algorille2.pdf.

[71] Frédéric Desprez et al. *Supporting Experimental Computer Science*. Tech. rep. 326. Argonne National Laboratory, 2012.

[72] Craig A. Stewart et al. “Jetstream: A Self-Provisioned, Scalable Science and Engineering Cloud Environment”. In: *Proceedings of the 2015 XSEDE Conference: Scientific Advancements Enabled by Enhanced Cyberinfrastructure*. XSEDE ’15. St. Louis, Missouri: Association for Computing Machinery, 2015. ISBN: 9781450337205.

[73] J. Mambretti, J. Chen, and F. Yeh. “Next Generation Clouds, the Chameleon Cloud Testbed, and Software Defined Networking (SDN)”. In: *2015 International Conference on Cloud Computing Research and Innovation (ICCCRI)*. 2015, pp. 73–79.

[74] NSF. *FutureGrid: An Experimental, High-Performance Grid Test-bed*. http://nsf.gov/awardsearch/showAward.do?AwardNumber=0910

[75] Lucas Nussbaum. “Testbeds Support for Reproducible Research”. In: *Proceedings of the Reproducibility Workshop*. Reproducibility ’17. Los Angeles, CA, USA: Association for Computing Machinery, 2017, 24–26. ISBN: 9781450350600.

[76] Kai Diethelm. “The Limits of Reproducibility in Numerical Simulation”. In: *Computing in Science and Engineering* 14.1 (2012), pp. 64–72.