UNIVERSITY OF CALIFORNIA

SANTA CRUZ

AGILE RESEARCH DELIVERY

A dissertation submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Ivo Jimenez

June 2019

The Dissertation of Ivo Jimenez
is approved:

Table of Contents

List of Tables **vii**

List of Figures **viii**

Abstract **xii**

Dedication **xiv**

Acknowledgments **xv**

1 Introduction **1**

1.1 Overview . . . 1

1.2 Contributions . . . 4

1.2.1 The Research Lifecycle . . . 4

1.2.2 Research Lifecycle Challenges Addressed by This Dissertation . . 9

2 Related Work **14**

2.1 Ad-hoc Human Workflows . . . 14

2.2 Source Code Repositories . . . 14

2.3 Data Repositories . . . 15

2.4 Experiment Repositories . . . 15

2.5 Virtual Machines . . . 15

2.6 Experiment Packing . . . 16

2.7 Ad-hoc Validation . . . 16

2.8 High-level Guidelines . . . 17

2.9 Scientific Workflow Engines 17

2.10 Goals for a New Methodology 17

3 Popper: A DevOps Approach to Addressing Research Lifecycle Challenges 19

3.1 The DevOps Methodology 20

3.1.1 DevOps Principles 22

3.1.2 The DevOps Toolkit 28

3.2 SciOps: DevOps Principles Applied to The Research Delivery Process . 32

3.2.1 Research Delivery: The Research Lifecycle Viewed Through The Software Delivery Glass 33

3.2.2 Applying DevOps Principles To Research Delivery 34

3.2.3 SciOps Principles 36

3.2.4 The SciOps Toolkit 38

3.3 The Popper Experimentation Protocol 40

3.3.1 Pipeline Folder Structure 41

3.3.2 Popper Compliance: Self-verifiable Experimentation Pipelines . . 42

3.4 Agility in Research Delivery 43

3.4.1 Creation 43

3.4.2 Dissemination 44

3.4.3 Validation 45

3.4.4 Curation 46

4 Reusable Tools For Addressing Research Lifecycle Challenges in Computer Systems Research 50

4.1 Fingerprinter: Lightweight Profiling of System Performance for Future Reference 51

4.1.1 Application- and Architecture-agnostic Performance Profiling . . 52

4.1.2 Obtaining Performance Fingerprints 55

4.1.3 Using Performance Fingerprints 56

4.1.4 Related Work 64

4.1.5 Future Work 64

4.2 Aver: Formally Specifying and Automatically Validating Scientific Claims 65
4.2.1 From Charts to Formal Scientific Claims 65
4.2.2 A First-order Logic-based DSL to Specify Scientific Claims . . . 67
4.2.3 Validation Engine 69
4.2.4 Automated Validation in Research Delivery 70
4.2.5 Related Work 70
4.2.6 Future Work 70
4.3 CLAPP: Cross-platform Validation of Scientific Claims 71
4.3.1 Inferred Performance Modeling 72
4.3.2 CLAPP pipelines 73
4.3.3 Evaluation 76
4.3.4 Future Work 82
4.4 quiho: Bottleneck Hints For Root Cause Analysis 85
4.4.1 Intuition Behind *quiho* 87
4.4.2 System Resource Utilization Via Feature Importance in SRA . . 88
4.4.3 Evaluation 92
4.4.4 Related Work 95
4.4.5 Future Work 96
4.5 CONFIRM: Statistically Sound Experiments on Bare-metal-as-a-Service Infrastructures 97
4.5.1 Variability in Systems Experiments 97
4.5.2 A Dataset For Building Statistical Tools 99
4.5.3 CONFIRM: A Tool for Designing Statistically Sound Experiments 100
4.5.4 Related Work 101
4.5.5 Future Work 102

5 Black Swan: Ongoing and Future Work 104
5.1 Popper In Practice 105
5.2 Black Swan 106
5.3 Use cases 108
5.4 Challenges 109

Bibliography **110**

List of Tables

3.1 Comparison of tasks executed as part of a scientific exploration against those executed in software delivery projects. 33

4.1 List of stressors used in this dissertation, along with the categories assigned to them by `stress-ng`. Note that some stressors are part of multiple categories. 53

4.2 Table of runtime measurements for the experiment contained in Fig. 4.1. The header of the table determines the predicates available to create FOL statements. Every row in an instance of this schema (a table with values) is a fact. 66

4.3 Example Aver statements corresponding to claims made in papers appearing in systems research articles. Rows 1-4 correspond to articles appearing in VLDB [181], OSDI [182], NSDI [183], and HPDC [184], respectively. . 68

4.4 Table of machines used in this study, grouped by architecture. 78

4.5 Aggregated accuracy results for each pipeline, over all three machines and nine applications. 80

List of Figures

1.1 A research lifecycle workflow. 5

1.2 A generic experimentation pipeline in CDS. At each stage of this pipeline, researchers carry out activities such as planning and designing experiments; writing, compiling, testing, and executing code; creating and managing datasets; visualizing results; writing notes, documentation and manuscripts. 6

1.3 Diagram illustrating my contributions. 10

3.1 The phases of the software delivery process, with stakeholders denoted by callout ovals. To simplify our discussion, we only talk about Development and Operations, assuming QA tasks are shared among the two domains. 21

3.2 The stages of the software delivery process. The infinite loop shape denotes the fact that this is an iterative process. Software delivery pipelines implement this process by making use of the DevOps toolkit (presented in Section 3.1.2). 23

3.3 An example of how a software delivery pipeline is stored in version control system. A pipeline at a given time points to all its dependencies. 25

3.4 End-to-end automation brings all the principles together in a unified process. 26

3.5 The stages of the research delivery process. Research delivery pipelines implement this process by making use of the DevOps toolkit described in Section 3.1.2. 35

4.1 Reprinting Figure 8 from the original paper. The original caption reads: "*OSD write performance scales linearly with the size of the OSD cluster until the switch is saturated at 24 OSDs. CRUSH and hash performance improves when more PGs lower variance in OSD utilization.*" 59

4.2 Reproducing a scaled-down version of the original OSDI '06 scalability experiment. The y-axis represents average throughput, as seen by the client. They x-axis corresponds to the size of the cluster (in number of object storage devices (OSD). The square marker corresponds to the average of 10 executions. The line with triangle markers projects the original results to our setting. This projection is obtained by having the 58 MB/s divided by 2 (to reflect the doubled I/O operation of the current Ceph version), i.e. 24 MB/s as the scalability unit of the original experiment. 61

4.3 Showing the effect of replacing 4 hard drives with newer models. Old hard drives are the same used in the previous figure and correspond to a set of 10 year old 250GB Seagate Barracuda ES (ST3250620NS). New hard drives correspond to 500GB Western Digital Re (WD5003ABYZ) drives. Every data point corresponds to the average (and standard error) of 10 executions. 63

4.4 CLAPP's workflow for obtaining a cross-platform prediction model for an application. 72

4.5 Idealized example that illustrates the concept of inferred performance modeling. In this example, we find that the performance of the application in question is directly correlated to the memory bandwidth of the underlying platform. With this information, we can create a linear model to represent the performance of this application, predicated on the underlying memory bandwidth metric. 74

4.6 Matrix of performance feature vectors over a collection of servers, along with a performance metric for an application. This matrix serves as the basis for building linear regression models for predicting the performance of applications on unseen architectures. To do this, the same set of microbenchmarks is used to obtain a performance feature vector for the unseen machine, and using this one can obtain the performance of one or more applications on this unseen machine. 75

4.7 Illustration on how the `maxtrials` parameter of `hyperopt` affects MAPE. 77

4.8 Accuracy of prediction models produced by the xgboost pipeline using `stress-ng` performance vectors. 79

4.9 Accuracy of prediction models produced by the xgboost pipeline using `likwid` performance vectors. 80

4.10 Accuracy of prediction models produced by the tensorflow pipeline using `stress-ng` performance vectors. 81

4.11 Accuracy of prediction models produced by the tensorflow pipeline using `likwid` performance vectors. 82

4.12 Variability of MAPE as the number of machines considered to build a model increases. 83

4.13 Variability of training loss as the number of machines considered to build a model increases. 84

4.14 *quiho*'s workflow for generating inferred resource utilization profiles (IRUPs) for an application. An IRUP is used as an alternative for profiling application performance and can complement automated regression testing. For example, after a change in the runtime of an application has been detected across two revisions of the code base, an IRUP can be obtained in order to determine whether this change is significant. IRUPs can also aid in root cause analysis. 86

4.15 An example profile showing the relative importance of features for an execution of the `hpccg` miniapp [207]. The x-axis corresponds to the relative performance value, normalized with respect to the most important feature, which corresponds to the first one on the y-axis (from top to bottom). Chapter **??** describes in detail how feature importances are calculated. 89

4.16 Automated regression testing pipeline integrating inferred resource utilization profiles (IRUP). IRUPs are obtained by *quiho* and can be used both, for identifying regressions, and to aid in the quest for finding the root cause of a regression. 92

4.17 IRUPs for the four tests benchmarked in this section. This and subsequent figures show only the top 5 most important features in order to improve visualization of the plots. 94

5.1 Black Swan will leverage existing services. 106

Abstract

Agile Research Delivery

by

Ivo Jimenez

Reproducibility is the cornerstone of the scientific method. Yet, in the computational and data-intensive branches of science, a gap exists between current practices and the ideal of having every new scientific discovery be *easily* reproducible. At the root of this problem are the dysfunctional forms of communication between the distinct stakeholders of science: researchers, their peers, students, librarians and other consumers of research outcomes working in ad-hoc ways; groups of individuals organized as independent silos, sharing minimal information between them, all of them with the common task of publishing, obtaining, re-executing and validating experimentation pipelines associated to scientific claims contained in scholarly articles and technical reports. This dissertation characterizes the practical challenges associated to the research lifecycle (creation, dissemination, validation, curation and re-use of scientific explorations) and draws analogies with similar problems experimented by software engineering communities in the early 2000's. DevOps, the state-of-the-art software delivery methodology followed by companies and open source communities, appeared in late 2000's and addresses these analogous issues.

By framing the problem of research delivery (iterating the research lifecycle) as a problem of software delivery, it becomes possible to repurpose the DevOps methodology to address the practical challenges faced by experimenters across the domains of computational and data-intensive science. This thesis presents Popper, an experimentation protocol for writing articles and carrying out scientific explorations following DevOps principles. Popper brings agility to research delivery in a domain-agnostic way by considering the end-to-end research delivery cycle, making it easier for all the stakeholders of science to publish, access, re-execute and validate experimentation pipelines. This dissertation also presents reusable tools in the domain of computer systems research. In a research

delivery context, this toolset covers the multiple phases of the research lifecycle and helps systems research practitioners carry out validations of scientific claims in this domain in an agile manner.

For this work, I was awarded the 2018 Better Scientific Software Fellowship, an initiative from the US Department of Energy whose goal is to foster best software development practices among the US scientific community. The Popper CLI tool, and its associated education materials, are currently being used to train new generations of scientists on how to be aware and guard against the multiple practical challenges in reproducibility.

This thesis is dedicated to the women in my life that have forged and continue to shape who I am: Rita, Ana Dolores, Ana Lucia and Eva. And to my brother and best friend Enrique.

Acknowledgments

I want to thank my advisor Carlos Maltzahn for all his guidance and support throughout my PhD studies. His focus on my person as a whole made this an enjoyable experience. I also want to thank my informal mentor Jay Lofstead for all his invaluable guidance in navigating the complex HPC world.

I want to thank Michael Sevilla and Noah Watkins: thank you for your friendship and disposition to collaborate with me, even though I was paying more attention to the frame than to the painting. I also thank all the members of the Systems Research Lab: Adam Crume, Andrew Shewmaker, Shel Finkelstein, Kathryn Dahlgren, Jeff LeFevre and Scott Brandt for their helpful suggestions and feedback. Many thanks to Lavinia Preston and Stephanie Lieggi for all their support.

I would also like to thank all my collaborators outside the SRL and UCSC: Kathryn Mohror, Adam Moody, Quincy Koziol, Sage Weil, Zack Cerza, Michael Berman, Ike Nassi, Peter Alvaro, Andrea Arpaci-Dusseau, Remzi Arpaci-Dusseau, Rob Ricci, Dmitry Duplyakin, Alexander Maricq, Josh Hacker, John Exby, Kevin Tyle, Ollie Lo, Pascal Grosset, Philip Kufeldt, and Katia Obraczka. Thank you for giving me the opportunity to learn from your experience.

This thesis work was partially funded by the Center for Research in Open Source Software, Sandia National Laboratories, NSF Awards OAC-1450488 and OAC-1836650, DOE Award DE-SC0016074. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Chapter 1

Introduction

1.1 Overview

Reproducibility is the cornerstone of the scientific method. Yet, in the **C**omputational and **D**ata-intensive branches of **S**cience [1] (CDS), a gap exists between current practices and the ideal of having every new scientific discovery be *easily* reproducible [2]. One of the main challenges in experimental CDS is the lack of stability induced by the continual stream of changes in software and hardware that make it difficult to run experiments in a reliable way. In this scenario of continual change, repeating an experiment that someone else has published with the expectation of obtaining the same results is commonly perceived as unrealistic, especially in computer systems research where performance is the main subject of study [3]. Additionally, advances in computer science (CS) and software engineering (SE) that address the challenges of continuously changing environments slowly and painfully make their way into the lifecycle of the different domains of CDIS—even in CS research itself, paradoxically [4–6].

The problem of dealing with continual change has been known to the SE community since its inception as a discipline [7] and, not surprisingly, has attempted to address it by coming up with methodologies, technologies and best practices that make this problem tractable [8–10]. The DevOps methodology [11–13] is the most recent incarnation in the evolutionary line of software delivery practices. The term *software delivery* [14–16] is used instead of *software development* in order to emphasize the fact that successful management of the complete application lifecycle is the main driver of success in a software-based company. Software delivery goes well beyond writing and testing code in a constrained development environment, it encompasses tasks such as infrastructure management, software deployment, and real-time monitoring.

The term *DevOps* is a clipped compound of the words “Development” and “Operations”, and denotes the combination of software development (“Dev”) with IT operations (“Ops”) as a methodology to address the problem of software delivery. The precise definition of DevOps that I use throughout this thesis is the one from [12]: “a set of practices intended to reduce the time between committing a change to a system and the change being placed into normal production, while ensuring high quality”. Through a bast array of tools and their underlying concepts such as version-control, hash-based namespaces, hardware- and OS-level virtualization, continuous integration, continuous deployment, and continuous monitoring, DevOps addresses the complexities of communication across the three main stakeholder domains that are in charge of the software application lifecycle: development, quality assurance and operations [17]. In short, DevOps is the state-of-the-art methodology that software engineers follow in order to test, release, deploy, monitor and maintain applications.

The success of DevOps outside of software engineering circles is understated. From small organizations all the way to Fortune 1000 companies, the DevOps methodology is transforming the way in which software products get delivered [18]. Even in highly-regulated industries such as finance and health, DevOps adoption is growing as well [19,20]. Analysts estimate that the market size of companies developing tools or providing IT services around DevOps will be 12.25 billion USD by the year 2025 [21]. GitLab, a prominent example of an open-source company within the DevOps space was recently valued in 1.1 billion USD [22].

And what does DevOps have to do with science? The goal of my work is to cast the research lifecycle in CDS as a problem of “research delivery”, that is, to view the research lifecycle as an instance of the problem of software delivery, in particular through the lens of DevOps. People across the disciplines of CDS spend their days carrying out activities associated to the research lifecycle: creation, dissemination, validation, curation and re-use of experiments. Students and researchers deal constantly with practical issues associated with these activities; issues that can be addressed by applying DevOps principles in a scientific context.

> **My thesis is that by understanding the practical challenges faced throughout the research lifecycle, it becomes possible to apply DevOps principles to accelerate the pace of scientific discovery.**

This dissertation describes challenges that practitioners face during the research lifecycle; introduces an experimentation protocol that holistically addresses the practical aspects of the research lifecycle by applying DevOps principles; and, within the domain of computer systems research, presents five tools that allow researchers to iterate the research lifecycle faster.

The rest of this dissertation is organized as follows:

- Section 1.2 presents a research lifecycle workflow and describes the practical challenges faced throughout it. I use this workflow to describe the analogies between software delivery (addressing application lifecycle issues) and research delivery (addressing research lifecycle issues). In addition, I also use this workflow to structure and describe my contributions.
- Chapter 2 surveys related work from the point of view of the research lifecycle, both across domains of CDIS and within computer systems research.
- Chapter 3 presents Popper [23–25], a domain-agnostic methodology for creating experimentation pipelines following DevOps software delivery principles. To the best of my knowledge, Popper is the first attempt at holistically addressing the problems associated to the research lifecycle by framing it from the point of view of software delivery. I give a brief introduction in Section 1.2.2.
- Chapter 4 presents reusable tools that address practical challenges found in the lifecycle of computer systems research. I give a brief introduction to these tools in Section 1.2.2.
- Chapter 5 outlines our ongoing and future work. This chapter presents Black Swan [26], our vision for an open-source, collaborative research delivery platform based

on Popper that allows researchers to integrate domain-specific tools such as the ones presented in Chapter 4, while at the same time allowing multiple communities of CDS to contribute to the domain-agnostic aspects of research delivery.

1.2 Contributions

People across the CDS disciplines spend their days carrying out activities associated to the research lifecycle: creation, dissemination, validation, curation and re-use of experiments. I now present (Section 1.2.1) a research lifecycle workflow [27–29] as a way of structuring the practical challenges faced while carrying out research in CDS. I use it throughout this thesis to describe the analogies between software delivery (addressing application lifecycle issues) and research delivery (addressing research lifecycle issues). I also use this workflow to structure my contributions (Section 1.2.2).

1.2.1 The Research Lifecycle

The diagram in Fig. 1.1 illustrates the research lifecycle workflow. The main phases are denoted by dashed-line boxes, while activities carried out at each phase are enclosed in solid-line boxes; main stakeholders at each phase are within call-out ovals.

The following subsections (1.2.1.1-1.2.1.5) describe in detail each of the main phases of the research lifecycle and the practical challenges faced while carrying them out. For each phase, I classify challenges in two: (1) those that are shared across disciplines of CDS and (2) those that are domain-specific, with an emphasis on issues faced in computer systems research. Lastly, subsection 1.2.1.6 discusses research lifecycle methodologies.

1.2.1.1 Creation

The core of the research lifecycle is the creation phase, where an experimentation pipeline is implemented and tested by its authors. Figure 4.16 shows a diagram of a generic pipeline typical of CDS. At each stage of this pipeline, researchers carry out activities such as planning and designing experiments; writing, compiling, testing and executing

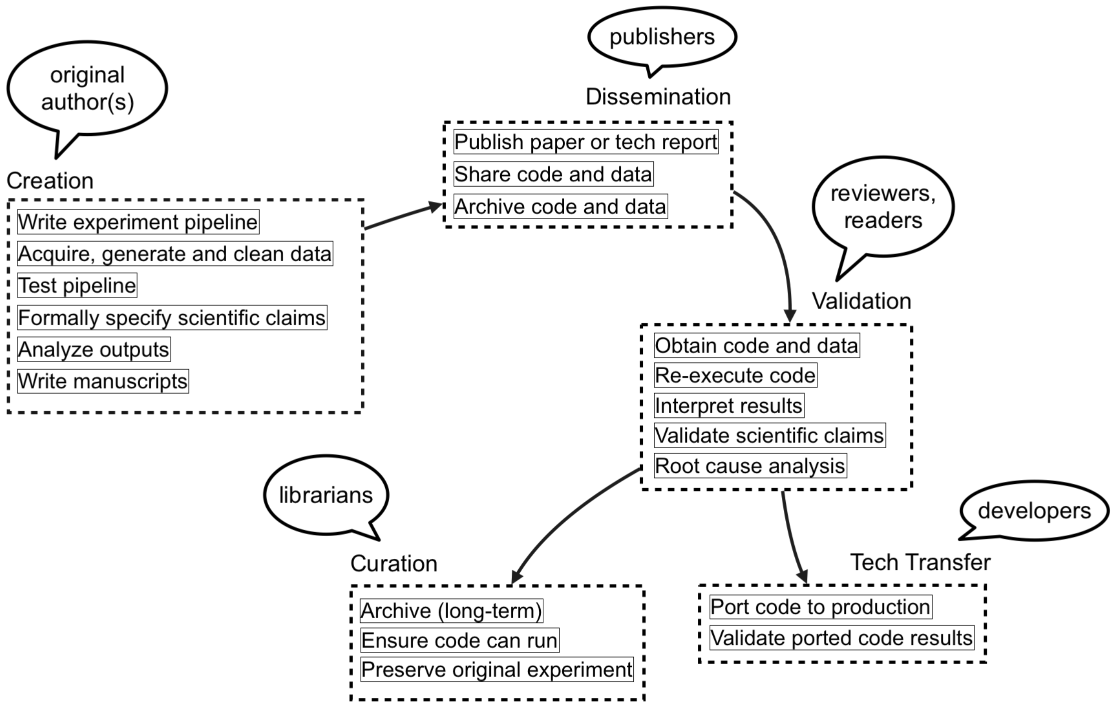


Figure 1.1: A research lifecycle workflow.

code; creating and managing datasets; visualizing results; writing notes, documentation and manuscripts in the form of technical reports and scholarly articles.

The set of domain-agnostic challenges in this phase is a proper subset of SE [30]: programming productivity [31,32], software testing [33,34], among others. Current practices for carrying out activities in this phase are ad-hoc and manual; little to no automation is implemented in CDS experimentation pipelines. One of the most challenging tasks (if not the most challenging) is the verification of experiments [35–37], i.e. ensuring that results produced by an experiment properly back the scientific claim being made.

In the domain of systems research, where performance is the main subject of study, the main challenge is to identify, understand and control the distinct sources of performance variability [3,38,39].

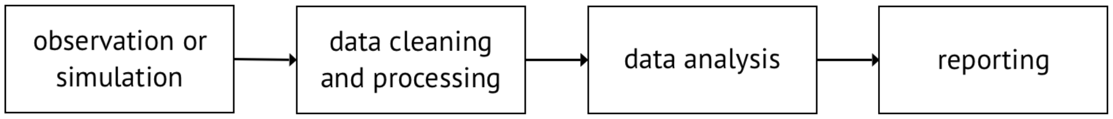


Figure 1.2: A generic experimentation pipeline in CDS. At each stage of this pipeline, researchers carry out activities such as planning and designing experiments; writing, compiling, testing, and executing code; creating and managing datasets; visualizing results; writing notes, documentation and manuscripts.

> **Goal**: Create and verify experimentation pipelines.
>
> **Challenges**: Increase productivity without impacting validity of results; ensure that an experimentation pipeline is properly backing up a scientific claim.

1.2.1.2 Dissemination

The dissemination phase is where scientific evidence and research insights get published and shared with others. From a practical point of view, this phase is where experiment code and data is shared with others. The most widely used form of communication is written manuscripts such as scholarly publications, internal technical reports or presentations.

The main challenge across CDS domains is to find the mechanisms to ensure that others can easily access all the assets associated to a scientific claim, including the complete experimentation pipelines as well as the associated context needed to execute them including inputs and output datasets [40–42]. Sharing all this information is necessary in order to properly validate claims.

Within the domain of computer systems research, the challenge is to disseminate the scholarly product in a format that captures the entire system context [6], such as compilation flags, parameters to subsystems of the BIOS and OS, among others.

> **Goal**: Publish and share experimentation pipelines.
>
> **Challenges**: What should be shared? In which format? How should it be organized? How should it be shared?

1.2.1.3 Validation

The validation phase is where others corroborate the claims made by the original authors. Several challenges arise in this phase across all domains of CDS, mainly in terms of re-executing an experimentation pipeline (prepare the computational environment), as well as interpreting results and determining whether original claims hold after re-executing it. Ideally, all this would be done *automatically and without requiring the intervention of the original authors.* The generally held assumption from researchers in CDS is that this phase is expensive to carry out both in time and effort.

Within systems research, some scientific claims rely on highly specialized conditions such as executing on custom hardware, proprietary software or running at very large scales [43]. Reviewers and readers might not have access to these resources due to security, economic or intellectual property issues, making it difficult to properly recreate the environment where an experimentation pipeline originally executed [44]. Even if hardware and software is the same, multi-tenancy [45], platform noise [46], non-deterministic ordering in parallel executions [47], among others, increase the complexity of this problem.

> **Goal**: Check if claims hold on new scenarios.
>
> **Challenges**: How can reviewers and readers re-execute an experimentation pipeline without the help of the original authors? How should results be interpreted? How do readers know that they have obtained results that back a scientific claim? If a claim does not hold? how should the root cause be investigated?

1.2.1.4 Curation

Curation is the phase where long-term maintenance of an experimentation pipeline is performed. Traditional curation activities involve cataloging, indexing and archiving research artifacts so they can be easily accessed [48]. In education, curated experiments are an invaluable teaching tool for introducing students to a new domain [49].

High-level goals such as the ones expressed by the FAIR principles [50] (findability, accessibility, interoperability and re-usability) frame the practical challenges in the curation of experimentation pipelines and datasets. Given the continually-changing nature of cyber infrastructure, one of the main challenges is to preserve the original context in which a pipeline was executed.

In systems research, there is a strong correlation between the results of an experimentation pipeline and the temporal dimension of technology. By temporal dimension, we mean the timeline of hardware technology, with a whisker in it determining the available hardware at that point in time, which in turn determines the performance behavior of a system. The claims contained in a systems research paper highly depend on its corresponding point in this timeline. The outcome of an experiment will certainly be different in the future due to the fact that new technology observes distinct performance behavior. A prominent example of this is the effect that solid-state drive (SSD) technology had on well-established performance behavior assumptions of relational databases, which up to that point assumed hard-disk drives (HDD) as the main storage medium [51,52]. Thus, in order to preserve an experiment as it was originally published, we need to capture and preserve the "performance landscape" of distinct system resources (memory, CPU, I/O, network) at the time a systems research experimentation pipeline is disseminated.

> **Goal**: Maintaining the functional integrity of a pipeline, regardless of whether results are correct or not.
>
> **Challenges**: How can long-term availability and reusability of code and data be guaranteed? How can the original context of an experiment be preserved so that claims are corroborated in exactly the same original conditions?

1.2.1.5 Technology Transfer

The technology transfer phase is where research insights are transferred to an operational or commercial environment [53,54]. The main challenge across all domains is validating that the same results obtained in the research environment hold when the technology is transferred to a production environment [55].

> **Goal**: Productize research insights, or incorporate them into an existing software product.
>
> **Challenges**: How can transfer time be minimized? How can original experiments be used to validate that the transfer has been done successfully?

1.2.1.6 Research Lifecycle Methodologies

In the context of this dissertation, a methodology is an outlining of the way in which experiments are implemented, disseminated, validated, curated and transferred, i.e. an outlining of the way in which experiments should be implemented and how they are communicated throughout the research lifecycle. This can be expressed in the form of a list of high-level guidelines [44,56], best practices [57–60], conventions [61] or formal protocols [62–64].

One of the biggest challenges throughout the research lifecycle is communicating what an author has done across all the lifecycle phases. Going across "stakeholder domains" usually comes with an associated loss of contextual information that was not properly communicated from the previous phase. For example, most of papers in systems research make the code of the system being studied available through Github or similar version-control web services. Important information such as how the code was compiled or on which platforms was executed is not made available. Validating claims without this information is impossible.

> **Goal**: Define methodologies for implementing experimentation pipelines that allow researchers to communicate all the necessary information for their proper consumption.
>
> **Challenges**: Can we define a domain-agnostic methodology? Can a methodology address the needs of all the stakeholders across the research lifecycle?

1.2.2 Research Lifecycle Challenges Addressed by This Dissertation

My contributions can be placed in the context of the research cycle workflow presented previously. In Fig. 1.3, I re-use the workflow shown in Fig. 1.1 and overlay symbols that denote the parts of the research delivery cycle that are addressed by my contributions[1]. These contributions can be organized in two broad categories: those that are domain-agnostic (Chapter 3) and those that pertain to the domain of computer systems research (Chapter 4).

[1]The boxes without a symbol are outside of the scope of my work, in particular, the tech transfer phase of the workflow which entails re-implementing code in other contexts.

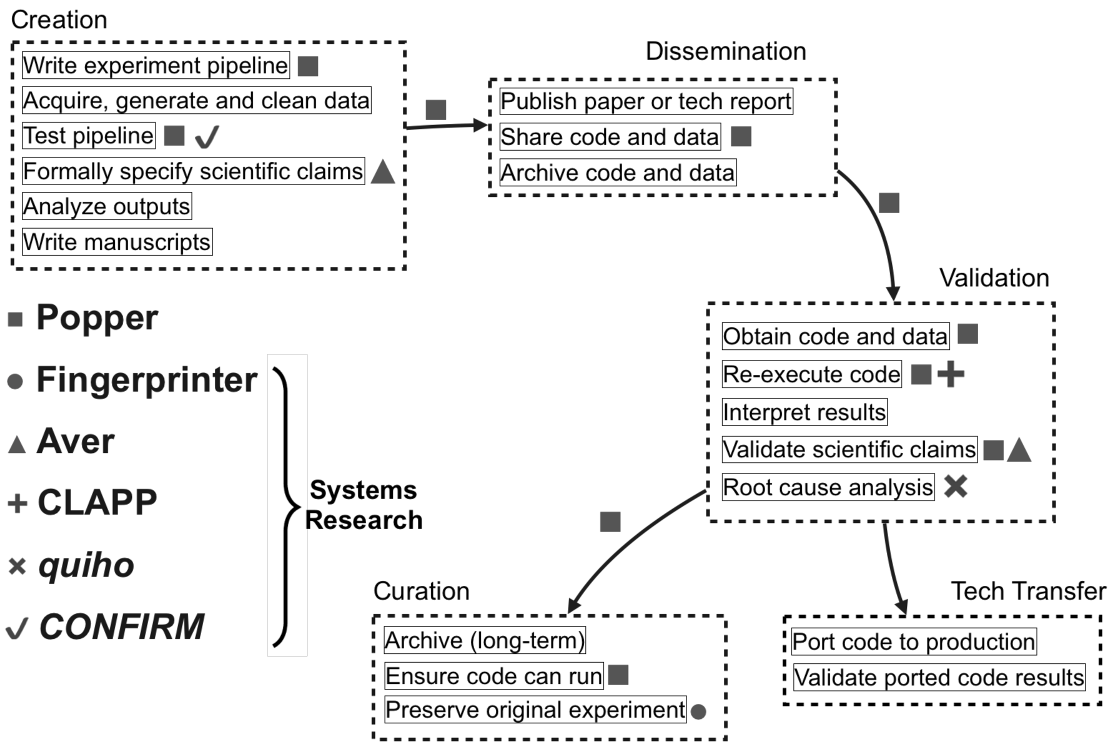


Figure 1.3: Diagram illustrating my contributions.

Popper [23–25] (Chapter 3) is a domain-agnostic experimentation protocol for creating experimentation pipelines following state-of-the-art software delivery principles (DevOps). The research lifecycle workflow presented in Fig. 1.1 defines a process: communicating research insights to the distinct stakeholders in research (reviewers, students, librarians, and software developers). This process involves writing and testing experimentation pipelines associated to scientific explorations. The problem of research delivery can be defined as the problem of iterating the end-to-end research delivery lifecycle as fast as possible without compromising scientific rigor. In this view, DevOps principles (that address the analogous problem of software delivery) can be adapted and extended to address the challenges that arise when one tries to bring agility to research delivery. The Popper protocol aids practitioners in applying DevOps principles, resulting in a unified, holistic approach to address the research lifecycle challenges described previously (1.2.1).

While the Popper protocol addresses the practical, domain-agnostic challenges faced by practitioners while carrying out the research lifecycle workflow, it is not sufficient in order to address all the issues in research delivery. Every discipline within CDS needs to address domain-specific aspects in order to avoid sacrificing scientific rigour. In this domain-specific context, having tools that aid in the creation, validation and curation of scientific claims has a direct impact on how agile can practitioners be while iterating the research delivery cycle. My contributions in the domain of computer systems research are listed below. They have the collective goal of bringing agility to the iteration of the research delivery cycle in systems research.

- **Fingerprinter** (Section 4.1). Validating claims in systems research requires having access to contextual temporal information about the performance context in which an experimentation pipeline is executed. Fingerprinter implements a lightweight performance profiling technique to characterize the underlying system where a pipeline runs [65]. This information can then be associated to executions of an experimentation pipeline so that it can later be used in the validation and curation of system research experiments. Since these profiles capture the performance landscape, they address the temporal issues in curation described in Chapter 1.2.1.4. In addition, these profiles can also be used as the basis for creating other tools, as is the case for CLAPP and *quiho* and CONFIRM (see below).

- **Aver** (Section 4.2). Currently, scientific claims contained in a paper are expressed in narrative form, which implies that a human needs to interpret the text in order to verify that the result of an experiment properly backs a scientific claim. In my work, I look at how claims can be formally specified in machine-readable format such that they can be automatically verified. Aver is a DSL for formally specifying (in first-order logic) scientific claims that refer to the outcome of an experimentation pipeline [66]. Aver statements are written by authors (creation phase) to express their claims and make them part of the experimentation pipelines. Aver is also the name of the validation engine that checks Aver statements against the output of a pipeline in order to validate scientific claims (validation phase).

- **CLAPP** (Section 4.3). In a scenario where an experimental infrastructure gets updated by adding a new batch of compute, network and storage devices to its inventory of machines, testing every existing pipeline on the new hardware represents a challenge since one would rather skip experiments for which we expect results to hold. In other words, it is desirable to allocate resources to test claims for which validations are likely to *not* hold, so that we can proceed to explain the root cause, which in turn might result in generating new research insights. **C**ross-platform **L**earning-**A**ssisted **P**erformance **P**rediction (CLAPP) is a tool for predicting performance regressions on newer, unseen hardware platforms [67]. CLAPP can be used to predict when an expectation will not hold, without having to re-execute an experiment (validation phase). When a pipeline has codified validation (e.g. using Aver statements), CLAPP can be used to automatically identify (filter) experimentation pipelines whose claims will not hold when executed on newer platforms, which can serve as scalable way of testing claims for repositories containing a large number of system research experiments.

- ***quiho*** (Section 4.4). In some cases, re-executing an experimentation pipeline will result in having codified expectations, such as the ones expressed with Aver, "break". That is, codified validations might not hold after an experiment has been re-executed. In systems research, from the point of view of performance engineering, this can can be seen as a performance regression. Learning-assisted profiling of resource utilization behavior can aid in root cause analysis of performance regressions [68] (validation phase). ***quiho*** assists the user in identifying the starting points of a root cause investigation.

- **CONFIRM** (Section 4.5). In systems research, a gap exists between current experimentation practices and the statistically sound analysis of experimental results that is followed in other domains of empirical research [3]. CONFIRM is a tool for sanitizing computer infrastructure in bare-metal-as-a-service platforms [69]. CONFIRM can be used in the creation phase of the research lifecycle by authors to filter out unrepresentative nodes using statistical tests as the elimination criteria (e.g. "only allocate nodes whose I/O performance is representative of that type of node with 95% confidence"). In addition, CONFIRM can be used prior to the execution of a pipeline in order to determine the number of repetitions that an experiment should be executed in order to obtain results within a user-provided confidence interval.

Chapter 2

Related Work

This chapter discusses existing approaches to addressing the challenges outlined in Chapter 1.2.1.

2.1 Ad-hoc Human Workflows

A typical practice is the use of custom bash scripts to automate some of the tasks of executing experiments and analyzing results. From the point of view of researchers, having an ad-hoc framework results in more efficient use of their time, or at least that is their belief. Since these are personalized scripts, they usually hard-code many of the parameters or paths to files in a local machine. Worst of all, important contextual information is in the mind of researchers rather than in the code. Without a list of guiding principles, going back to an experiment, *even for the original author, on the same environment*, represents a time-consuming task.

2.2 Source Code Repositories

Version-control systems [70–72] give authors, reviewers and readers access to the same code base but the availability of source code does not guarantee reproducibility [6]. While sharing source code is beneficial, it leaves readers with the daunting task of recompiling, reconfiguring, deploying and re-executing an experiment. Things like compilation flags, experiment parameters and results are fundamental contextual information for re-executing an experiment that is missing.

2.3 Data Repositories

A data repository [73–76] is often used as an alternative to source code repositories. These are accessed through a Web UI and are treated as a deposit of all files associated to a publication. Since they are not source code repositories, they do not keep track of provenance and do not provide versioning capabilities, so what gets deposited is a snapshot of what a researcher's code and data looked like at the time they were uploaded. Similarly to source code repositories, the lack of a common folder structure and formats makes it difficult for readers to easily re-execute a published experiment.

2.4 Experiment Repositories

Experiment repositories [77–79] allow researchers to upload artifacts associated with a paper. Similar to code and data repositories, one of the main problems is the lack of automation and structure for the code that gets uploaded. The availability of the artifacts does not guarantee the reproduction of results since a significant amount of manual work needs to be done after these have been downloaded. Additionally, large data dependencies cannot be uploaded since there is usually a limit on the artifact file size.

2.5 Virtual Machines

Hardware virtualization [80–83] can be used to partially address the limitations of sharing source code. A virtual machine (VM) image can be used as a container of the complete software stack, from the OS up, along with any data dependency inside a VM image. However, in the case of systems research where the performance is the subject of study, the overheads in terms of performance (the hypervisor “tax”) and management (creating, storing and transferring) can be high [84] and, in some cases, they cannot be accounted for easily [85]. In scenarios where OS-level virtualization [86] is a viable alternative, it can be used instead of hardware-level virtualization to pack all the software dependencies of an experiment [87]. In addition, neither containers nor VMs are suitable for dealing with large datasets.

2.6 Experiment Packing

Experiment packing entails tracing an experiment at runtime to capture all its dependencies and generating a package that can be shared with others [88–93]. In addition, since this packages are created by tracing system calls that an experiment do to the OS, this approach can also be used to create provenance graphs [94]. The resulting packaged experiment is a compressed folder that can be shared with others so they can re-execute the exact same experiment. External dependencies such as large datasets cannot be packaged; the experiment is a black-box without contextual information (e.g. history of modifications) that is difficult to introspect and to build upon; and packaging does not explicitly capture validation criteria.

2.7 Ad-hoc Validation

Assuming the reader is able to recreate the environment of an experiment, validating the outcome requires domain-specific expertise in order to determine the differences between original and recreated environments that might be the root cause of any discrepancies in the results. Additionally, reproducing experimental results when the context changes (hardware and software modifications) is challenging mainly due to the inability to predict the effects of such changes in the outcome of an experiment [3,38]. In this case validation is typically done by “eyeballing” figures and the description of experiments in a paper, a subjective task, based entirely on the intuition and expertise of domain-scientists.

2.8 High-level Guidelines

High-level guidelines [58,60,95]. Other intermediate approaches rely on the specification of high-level workflows [96] and ignore some of the details of how an experiment is precisely defined.

2.9 Scientific Workflow Engines

Scientific workflow engines [97] are a specialized form of a workflow management system designed specifically to compose and execute a series of computational or data manipulation steps, or workflow, in a scientific application. Taverna [64] and Pegasus [98] are examples of widely used scientific workflow engines. In recent years, a plethora of domain-specific workflow engines have emerged[1].

2.10 Goals for a New Methodology

Current approaches surveyed previously partially address the challenges presented in Chapter 1.2.1. Thus, we see the need for a new methodology that:

- Is domain-agnostic: the same methodology applies to any discipline of CDS.
- Improves the personal workflows of scientists: a common methodology that works for as many distinct projects as possible.
- Captures the end-to-end research lifecycle: enables researchers to produce artifacts that can be used as the basis of communication and validation throughout the entire research lifecycle.
- Improves the efficiency of researchers: require the same or less effort than current practices with the difference of carrying out tasks in a systematic way.
- Maximizes the use of domain-agnostic tools (do not reinvent the wheel!).

[1] https://github.com/pditommaso/awesome-pipeline

- Captures validation criteria in an explicit manner: subjective evaluation of results of a re-execution is minimized.
- Results in experiments that can be easily shared and extended.
- Minimizes the involvement of original authors in the validation of scientific claims.

Chapter 3 presents Popper, a methodology that complies with all the criteria outlined above. Chapter 4 presents reusable tools that address challenges found in the lifecycle of computer systems research.

Chapter 3

Popper: A DevOps Approach to Addressing Research Lifecycle Challenges

In this chapter I introduce Popper, a protocol for the creation of experimentation pipelines based on DevOps principles.

- Section 3.1 introduces the DevOps methodology. First I describe the problem that DevOps solves by employing milestones in its history. I then give a concrete list of the principles underpinning the DevOps methodology and briefly describe the categories in which the DevOps tools can be classified on and show an example a software delivery pipeline.
- Section 3.2 describes how the research lifecycle introduced in Chapter 1.2.1 can be casted as a research delivery problem, and draw analogies with software delivery. I apply the DevOps principles to the research delivery process; I name this *SciOps* for short. I identify the need for narrowing and extending some of these principles in order to apply them in the scientific context and list the resulting SciOps principles.
- Section 3.3 introduces Popper, an experimentation protocol for implementing research delivery pipelines following SciOps principles.
- Section 3.4 concretizes all preceding sections by discussing how SciOps and Popper help researchers and students iterate the research lifecycle faster.

3.1 The DevOps Methodology

Software engineering (SE) methodologies consist of high-level guidelines that dictate how to go about writing software [99–101]. Methodologies have been proposed ever since the first programs were written, going back as early as 1960's according to Elliott et al. [102]. Fast-forwarding to the year 2001, a group of experts in software development methodologies (part of the OOPSLA community) met to discuss the challenges and possible solutions to what they thought was, at that time, a fundamental problem in software development [103]. In their view, traditional methodologies such as the Waterfall [104,105] and Rational Unified Process (RUP) [106] methods were out of touch with the realities of software delivery. In this type of outdated methodologies, the planning, coding, testing and release of software (application lifecycle) was done in "a single pass", with long release cycles (1-2 releases per year) and slow incorporation of customer feedback [107]. This group of experts published "The Manifesto for Agile Software Development" [108], a list of principles that encourage rapid and flexible response to change in the application development lifecycle. Instead of a single six-month cycle, they proposed to iterate as quickly as possible in order to get customer feedback early in the development of an application; hence the term "agile".

Arguably, the agile approach had a profound effect in software development practices [109–111]. Nowadays, SE practitioners are familiar with words such as Scrum [112] or Extreme Programming (XP) [113]. However, the manifesto was written before the era of internet web services, and these views were primarily motivated by issues found in a context where desktop applications and single-node web services were developed. During the first half of 2000s, internet companies experienced an exponential growth in their costumer base and established themselves as highly influential software development shops [114,115]. The web services landscape brought new challenges in software delivery that had never been experienced before: the need to be agile in a distributed computing scenario where issues such as reliability and scalability play a central role [116].

In those days of mid 2000s, the web service development environment was mainly comprised of two types of roles: developers and operators[1] (Fig. 3.1). Developers

[1]The study of DevOps can also be done by identifying three stakeholder domains in software delivery:

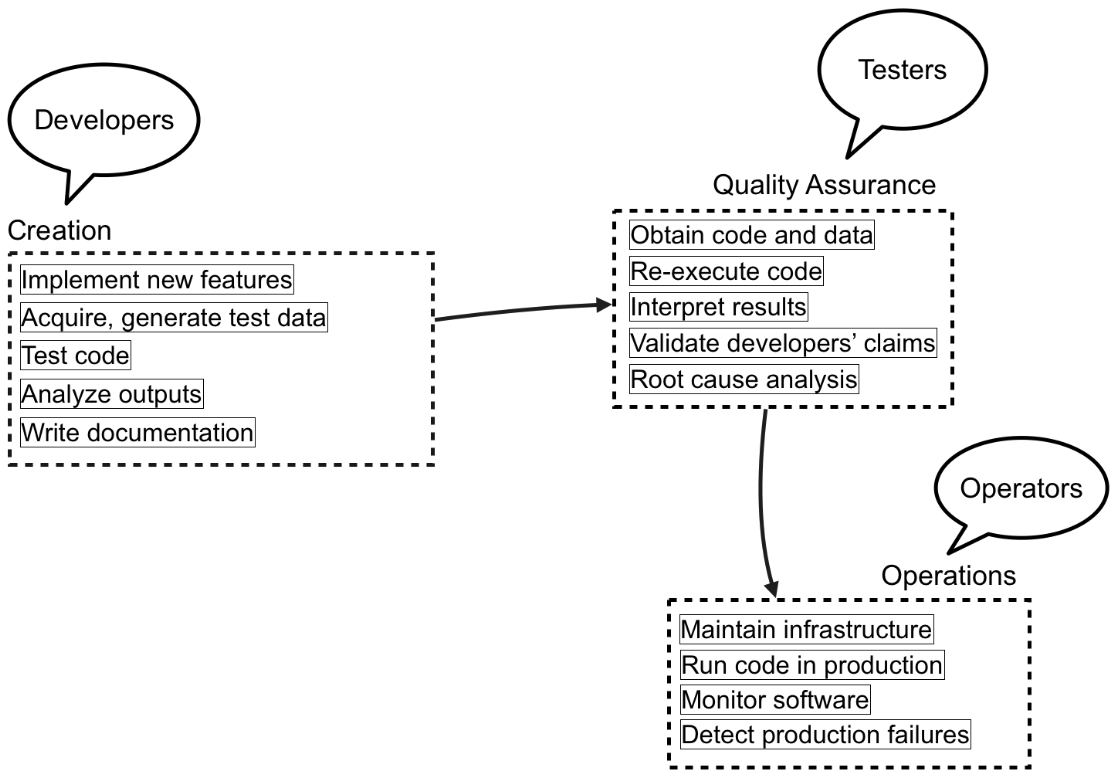


Figure 3.1: The phases of the software delivery process, with stakeholders denoted by callout ovals. To simplify our discussion, we only talk about Development and Operations, assuming QA tasks are shared among the two domains.

where in charge of implementing new features and ensuring that new changes did not break existing functionality of a set of distributed, web-based applications. Operators, also known as "sysadmins", were in charge of maintaining hardware and software infrastructure, deploying new versions of software, as well as ensuring that services are always up and running. These groups worked independently from each other and communicated in ad-hoc ways [117,118]. Having these two stakeholder domains organized as independent silos created contradicting goals: in order to address constantly-changing customer needs, developers need to frequently update an application's codebase; in order to maximize up-time of services, operators need to minimize the rate of change.

The dysfunctional communication patterns between these two siloed groups also generates

development, quality assurance (QA) and operations. To simplify our discussion, we assume QA tasks are shared among development and operations domains.

a lack of trust [119]: developers think operators do not understand how software works (because they do not write it); operators think that developers do not understand the real problems of keeping services running (because they do not maintain infrastructure, nor deploy their software in production). In 2009, the term DevOps appeared; it was used to name “DevOps Days” [120], a conference whose purpose was to convene developers and operators to discuss new emergent technologies being used by a niche of OSS projects and within internet companies to deploy multiple versions of a applications per day without service disruption [121,122]. The goal of this new type of tools, and its nascent associated methodology, was to take an agile approach to the issue of software delivery in the era of web services [123]. Soon after, the term was adopted by a broader community and thus the DevOps methodology was born.

The main objective of DevOps is to “break the communication barriers” between developers and operators [119]. This is done by taking a holistic approach to the delivery of software, mainly by applying software engineering principles to the operations side of software delivery (Fig. 3.2). In the next section I list and describe in detail the principles underpinning the DevOps methodology. From the high-level point of view, one of the key aspects that DevOps enables is the close collaboration between developers, testers and operators by having a single team whose members have varying degrees of expertise on the multiple aspects of software delivery. Instead of seeing stakeholder domains (development, testing, operations) as independent silos of experts, DevOps views these as skills that one engineer can posses. An engineer might be more knowledgeable in infrastructure automation than another that is more knowledgeable on the internal architectural aspects of an application; both of them work in the same team, and know the basics of the entire delivery process[2].

3.1.1 DevOps Principles

By *principle*, I mean a rule that dictates what should be done. And I use the term *practice* to refer to the specific actions that follow a principle. In other words, a principle

[2]The term “fullstack developer” is a term that is commonly used to refer to a software engineer that is knowledgeable on the end-to-end aspects of software delivery and the associated tools.

[3]This image is published by Wikipedia under a CC BY-SA 4.0 license https://creativecommons.org/licenses/by-sa/4.0/.

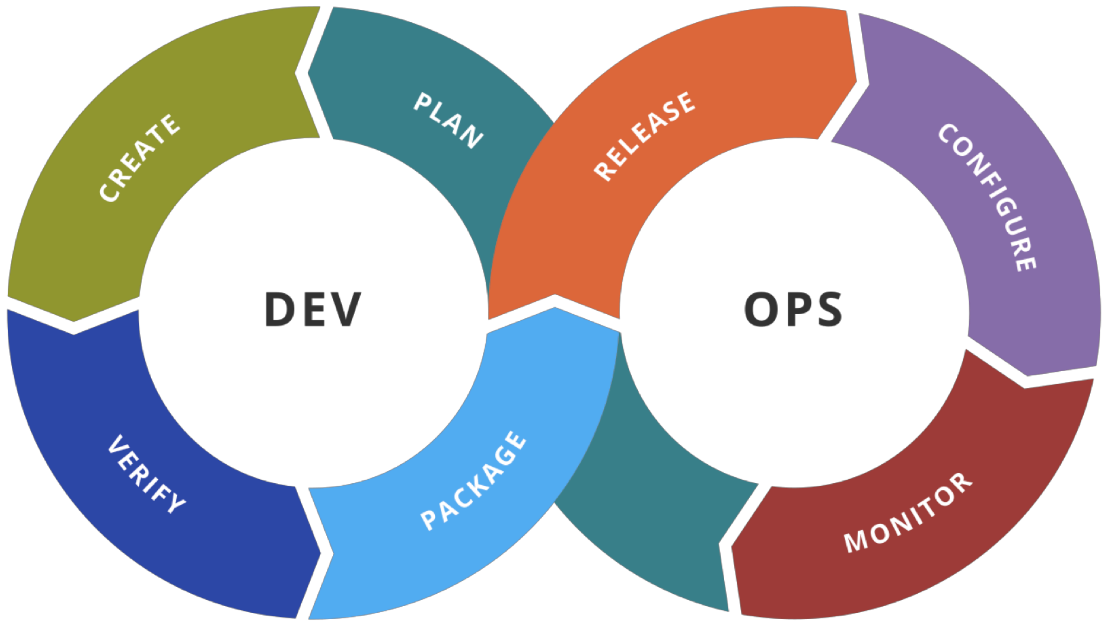


Figure 3.2: The stages of the software delivery process[3]. The infinite loop shape denotes the fact that this is an iterative process. Software delivery pipelines implement this process by making use of the DevOps toolkit (presented in Section 3.1.2).

dictates what to do, while a practice outlines how to do it. Thus, in this section I list the principles underpinning the DevOps methodology, which are implemented in distinct ways, using distinct tools (discussed in Section 3.1.2).

There are few academic resources on DevOps. Most of what I write in this section is spread throughout several references, as I was unable to find a "canonical" academic article that would list explicitly and in detail all the principles underpinning the DevOps methodology. All this is coming from a list of technical articles [17,123,124], books [11–14,119,125], blogs [16,121] and recorded talks [126–130].

3.1.1.1 Infrastructure-as-code

Specify the state of infrastructure in machine-readable format.

One of the iconic principles that DevOps embraces is the treatment of infrastructure as code [131,132]. This refers to the provisioning and management of compute and network resource by using files written in machine-readable format, usually written in plain-text formats such as YAML or JSON (see example in Tbl. **??**). These files specify the state and configuration of infrastructure that a developer assumes in order to ensure the correct behavior of an application. They also serve as the basis of collaboration and communication, with respect to environmental expectations that a team has on computational infrastructure. In practice, a written specification of what a team is expecting defines a common language to communicate between the development and operations domains, and also results in operations issues being addressed early in a software project. Also, being plain-text, these files are amenable to version-control (see 3.1.1.2 below).

3.1.1.2 Systematic change management

Manage changes in a systematic way by applying version-control principles.

By this principle, hash-based namespaces are used to assign identifiers to actionable objects in a software delivery pipeline [133]. Version-control systems a prominent example

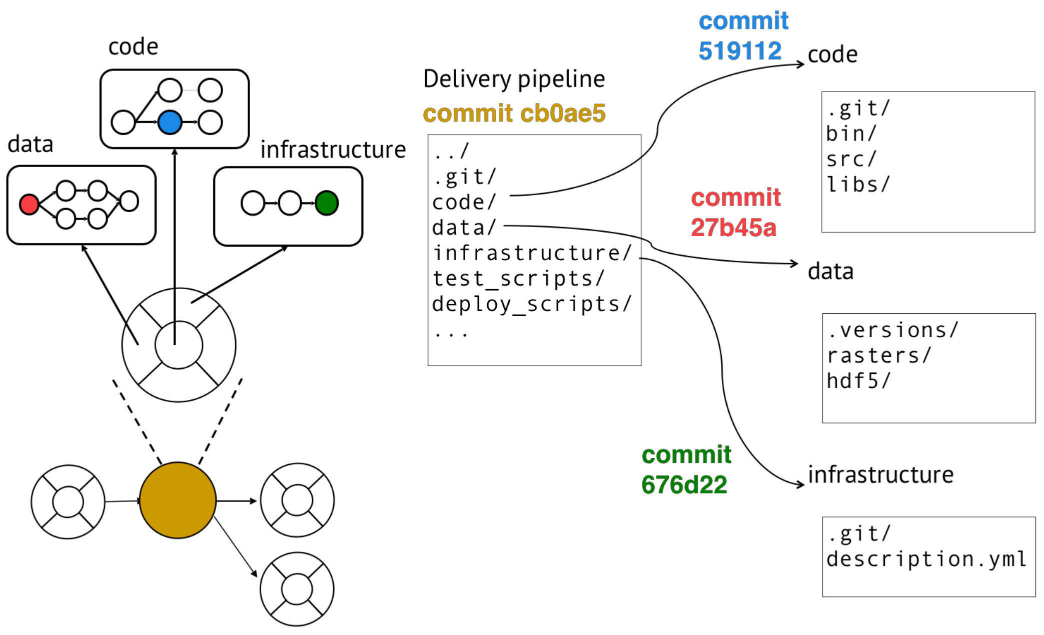


Figure 3.3: An example of how a software delivery pipeline is stored in version control system. A pipeline at a given time points to all its dependencies.

of this approach [134]. A persistent data structure [135,136] and content-addressable storage [137] is a common way of implementing a hash-based namespace. IDs can be used to identify anything from software releases to the (immutable) state of infrastructure [138]; every dependency in a software delivery pipeline can be accessed and systematically evolved by making use of these hash IDs (Fig. 3.3). More importantly, these IDs can be passed to automation tools so that they can act upon them. For example, a developer can request to run a specific version of tests on a particular version of the code base, using a particular infrastructure configuration. Having a hash-based namespace allows to overlay on top all the features of version-control systems such as tracking all changes (provenance) of objects in the namespace [139] (e.g. which VM image is the predecessor of the one running now [140]).

3.1.1.3 End-to-end automation

Automate the entire software delivery process.

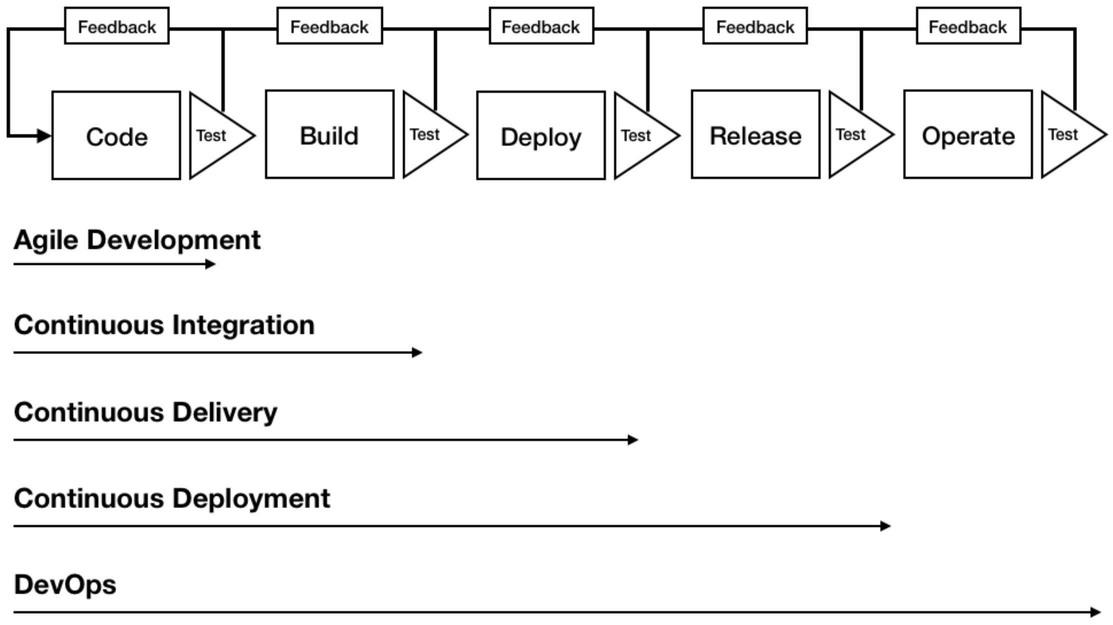


Figure 3.4: End-to-end automation brings all the principles together in a unified process.

Deploying multiple versions of a web service, along with all of its dependencies, while ensuring high quality cannot be done manually. One of the key principles of DevOps is to automate as much as possible the software delivery pipeline, which in turn reduces the occurrence of human-induced errors [141]. The diagram on Fig. 3.4 illustrates how an end-to-end pipeline looks like and how automation ties all the process and the principles (described below).

3.1.1.4 Self-service

Empower members of a team by allowing them to have access to all the assets associated to a software delivery pipeline.

Remove human bottlenecks by making the entire delivery pipeline readily available to anyone in the team [142]. In practice, this means that all the objects associated to a pipeline are stored in code and data repositories, with all members of a team having access to them. Additionally, the infrastructure where a pipeline runs is accessible to the team, so anyone in the team can run tests on-demand without having to wait for permissions, on the exact same infrastructure where the service runs in production. In order to avoid disrupting a service, a common practice is to define development, staging and production infrastructure areas.

3.1.1.5 Continuous integration

Continuously test the changes made to the codebase of an application.

Having end-to-end automation results in being able to trigger the execution of tests of a distributed application, by clicking a button [141]. This principle was originated in Agile software development practices and dictates that test should run as frequently as possible, ideally for every new commit that is done to a software delivery pipeline dependency. When the rate of change is too high to test on every commit, there are strategies to selectively execute tests [143].

3.1.1.6 Continuous deployment

Continuously deploy an application to production.

This principle is an extension of continuous integration and it dictates that deployment of an application should be done as frequently as possible, ideally on every new change to the code base. In order to minimize service disruption, a common practice is to perform an update following a rolling upgrade approach [144], where a service is updated on a small subset of target nodes first, and is expanded based on results of real-time monitoring (see 3.1.1.7 below).

3.1.1.7 Continuous monitoring

Continually monitor the behavior of an application.

This allows a team to continuously obtain runtime metrics of an application in order to detect malfunctions as early as possible [145]. This is not only related to functionality, performance or security, but also to ensure compliance with internal and external regulations [146].

3.1.1.8 Continuous feedback

Continually act on feedback generated by the continuous monitoring of an application.

This allows a team to get immediate feedback regarding unexpected behavior of an application [147]. This also refers to continuously gather feedback from users, not just infrastructure, and to process that feedback as quickly as possible and not in sprints, even automatically [148].

3.1.1.9 Toolchain homogeneity

Use the same tools across the distinct stakeholder domains.

Use the same set of tools to manage the entire software delivery pipeline [149]. Tools are a reflection of the way people collaborate and the methodology they follow; in turn, methodologies influence the processes that people follow [119]. This synergy between culture and tooling is evident in DevOps. Before DevOps, developers and operators, being isolated from each other, each used a distinct set of tools. For example, operators would create custom scripts to manage system infrastructure that were out of reach to developers. By collaborating closely together, the tools that both domains used converged into a single set, a result of the unification of software delivery process (Fig. 3.2). In the next section, I survey the toolkit by looking at the categories of tools, rather than individual tools.

3.1.2 The DevOps Toolkit

In this section I survey and highlight salient features of the DevOps toolset, organized by category. This toolkit is used to implement the DevOps principles outlined previously.

3.1.2.1 Version Control

Version-control systems systematically manage the changes done to a code base. More generally, any set of digital objects can have version-control as its main mechanism for managing changes of such objects. This category of tools are used to implement the "Systematic change management" principle, as well as the "Self-service" principle.

Tools and services: Git[4], Svn[5] and Mercurial[6] are popular VCS tools. GitHub[7], GitLab[8] and BitBucket[9] are web-based Git repository hosting services. They offer all of the distributed revision control and VCS functionality of Git as well as adding their own features. The entire history of the project and its artifacts of public repositories can be browsed online.

3.1.2.2 Package Management

The goal of package management is to systematically address the problem of dependencies between distinct pieces of software on a single system, thus implementing the "Systematic change management" principle.

Tools: Traditional package managers in Linux such as Apt, Yum and Apk. Alternatives are so called "modern package managers" such as Nix[10] or Spack[11].

3.1.2.3 Software Portability

While package managers allow to manage dependencies of an application, one still needs to address the issue of running the same code on multiple platforms. Software portability tools address this problem by interposing a virtualization layer between an application and the underlying system. This isolation layer can be at the level of the programming language runtime, Operating System or hardware.

Tools and services: At the level of the programming language Python (Virtualenv), R (packrat) and others help to isolate and recreate environment with ease. For projects that use more than one language or languages without isolation features, Docker[12] automates

[4]http://git-scm.com
[5]https://subversion.apache.org
[6]https://www.mercurial-scm.org
[7]http://github.com
[8]http://gitlab.com
[9]https://bitbucket.org
[10]https://nixos.org/nix/
[11]https://github.com/LLNL/spack
[12]http://docker.com

the deployment of applications inside software containers by providing an additional layer of abstraction and automation of operating-system-level virtualization on Linux. In the case of virtual machines, Vagrant[13]. Modern package managers can also deal with the issue of isolating environments.

3.1.2.4 Infrastructure Automation

This set of tools implement the principle of “Infrastructure-as-code”. They automate the task of allocating computational resources.

Tools and services: Cloudlab[14], Chameleon[15], PRObE[16] and XSEDE[17] are NSF-sponsored infrastructures that allows users to request computing resources to execute multi-node experiments. Additionally, public cloud service providers such as Amazon, Google, Packet and Rackspace allow users to deploy applications on virtual and bare-metal instances. Terraform[18] is a tool that allows to automate the configuration and provisioning of infrastructure in a platform-agnostic way. When a Terraform provider for a particular infrastructure is not available, one can resort to using platform-specific tools directly. For example CloudLab [150], Grid500K [151], AWS CloudFormation[19] and OpenStack Heat[20] provide CLI tools for this purpose.

3.1.2.5 Configuration Management and Environment Capture

This set of tools also implement the principle of “Infrastructure-as-code”. They automate the task of configuring computational resources and can also serve for capturing the details about the runtime environment (e.g. hardware and OS configuration, versions of system libraries, etc.). This type of tools implement the “End-to-end automation” and “Systematic change management” principles.

[13]http://vagrantup.com
[14]http://cloudlab.us
[15]http://chamaleoncloud.org
[16]https://www.nmc-probe.org
[17]http://xsede.org
[18]https://terraform.io
[19]https://aws.amazon.com/cloudformation
[20]https://wiki.openstack.org/wiki/Heat

Tools and services: Ansible[21] is a configuration management utility for configuring and managing computers, as well as deploying and orchestrating multi-node applications. Similar tools include Puppet[22], Chef[23], Salt[24], among others.

3.1.2.6 Dataset Management

Some software delivery pipelines deal with the processing of large datasets. While possible, traditional VCS tools such as Git were not designed to store large files. A proper artifact repository or dataset management tool is used to handle data dependencies. This type of tools help implement the "Systematic change management" principle since these tools provide versioning of the data objects they store.

Tools and services: Examples are Apache Archiva[25], Git-LFS[26], Datapackages[27] or Artifactory[28].

3.1.2.7 Continuous Integration and Deployment

This type of tools implement the "Continuous integration and continuous deployment" principles.

Tools and services: Travis CI[29] is an open-source, hosted, distributed continuous integration service used to build and test software projects hosted at GitHub. Alternatives to Travis CI are CircleCI[30] and CodeShip[31]. Other self-hosted solutions exist such as Jenkins[32].

[21] http://ansible.com
[22] https://puppet.com
[23] https://www.chef.io
[24] https://saltstack.com/salt-open-source
[25] https://archiva.apache.org
[26] https://www.nmc-probe.org
[27] http://frictionlessdata.io/data-packages/
[28] https://www.jfrog.com/artifactory
[29] https://travis-ci.org/
[30] https://circleci.com
[31] https://codeship.com
[32] http://jenkins-ci.org

3.1.2.8 Continuous Monitoring

Monitoring tools give access to runtime performance information. These tools can be used to implement the principles of “Continuous monitoring”. If the streams that are generated by this tools are analyzed in real-time, this in turn implements the principle of “Continuous feedback”.

Tools and services: Many mature monitoring tools exist such as Nagios[33], Ganglia[34], StatD[35], CollectD[36], among many others. For measuring single-machine baseline performance, tools like Conceptual[37] (network), stress-ng[38] (CPU, memory), fio[39] and many others exist.

3.2 SciOps: DevOps Principles Applied to The Research Delivery Process

The research lifecycle workflow presented in Fig. 1.1 defines a process: communicating research insights to the distinct stakeholders in research (reviewers, students, librarians, and software developers). This process involves writing and testing experimentation pipelines associated to scientific explorations. In the same way than software development is not the same as software delivery (Section 1), writing experimentation code as part of a research project is not the same as delivering those research insights. The problem of research delivery can be defined as the problem of iterating the end-to-end research delivery lifecycle as fast as possible without compromising scientific rigor.

[33]https://www.nagios.org
[34]http://ganglia.info
[35]https://github.com/etsy/statsd
[36]https://collectd.org
[37]https://github.com/lanl/coNCePTuaL
[38]http://kernel.ubuntu.com/~cking/stress-ng
[39]https://github.com/axboe/fio

3.2.1 Research Delivery: The Research Lifecycle Viewed Through The Software Delivery Glass

If we compare the research lifecycle workflow against the process of software delivery presented in the last section, we observe a close correspondence in terms of the tasks that a professional software engineer carries out as part of their job (Tbl. 3.1). Software engineers need to ensure that their software gets delivered correctly. To do so, they write and document tests, manage datasets, run multiple types of tests, and so on.

Table 3.1: Comparison of tasks executed as part of a scientific exploration against those executed in software delivery projects.

Scientific Exploration	**Software Project**
Write experiment code	Write tests
Manage datasets	Manage test cases
Share code and data	Package and release code and data
Analyze and visualize results	Analyze test results
Verify claims hold	Run unit, integration and non-functional tests
Validate claims in distinct scenarios	Ensure code runs OK in production
Write manuscripts	Document software

In academic scenarios, there are analogous problems to the ones that the DevOps methodology tries to address. As illustrated in the research lifecycle workflow (Fig. 1.1), distinct stakeholders try to get access to the outcome of researchers. While not a proper "operator" role, the role of a student, research peer or librarian is to get access to an experimentation pipeline that an author wrote and re-run it, either to validate claims, extend it so new research is generated, or curate it. Thus, in the scientific context, we see the same silos that are present in the software engineering world.

If we analyze this siloed academic culture, we can observe that the same dysfunctional communication patterns are present [152,153]. The main means of communication is a PDF file, while the actual scholarship is the entire experimentation pipeline, including all its dependencies and the associated context of where it ran [154]. The state of our practice is such that repeating an experiment that someone else has published with the expectation of obtaining the same results is commonly perceived as unrealistic [6], especially in computer systems research where performance is the main subject of study [3].

3.2.2 Applying DevOps Principles To Research Delivery

Can we apply DevOps principles to address the communication and lack of trust between silos found in research delivery? I now describe the outcome of going over this exercise. The phases of the research delivery cycle are presented in Fig. 3.5. We refer to this view of the research cycle through the lens of DevOps as SciOps[40] as a short form for "DevOps principles applied (and adapted) to the problem of research delivery". When contrasting DevOps and SciOps delivery processes, we see that DevOps focuses on software delivery pipelines; SciOps focuses on research delivery pipelines. The main difference is on what comes after we finish coding. DevOps deploys and continuously monitors an application; SciOps validates claims in new scenarios.

The nature of the tasks carried out in the creation phases of both SciOps and DevOps are similar; they mainly boil down to implementing a software delivery (DevOps) or research delivery (SciOps) pipeline. The differences appear when we contrast the "Ops" part of each. We can broadly speak of three main differentiator factors:

- **Validating a claim vs. deploying a service**. In software delivery, the goal is to have pipelines that result in deploying and continuously monitoring an application or service; in research delivery, the goal is to validate a claim [156,157]. At the mechanical level might look like a software delivery pipeline, but one does not expect to "deploy" an experiment. In other words, an experiment can be seen as a short-running application in relation to the type applications typically deployed by DevOps. One only runs a pipeline to prove a claim, not to execute the IT infrastructure backing up a business.

[40]The term was coined at the SC'17 BoF on Practical Reproducibility [155].
[41]This is an image modified from the one published by Wikipedia under a CC BY-SA 4.0 license https://creativecommons.org/licenses/by-sa/4.0/.

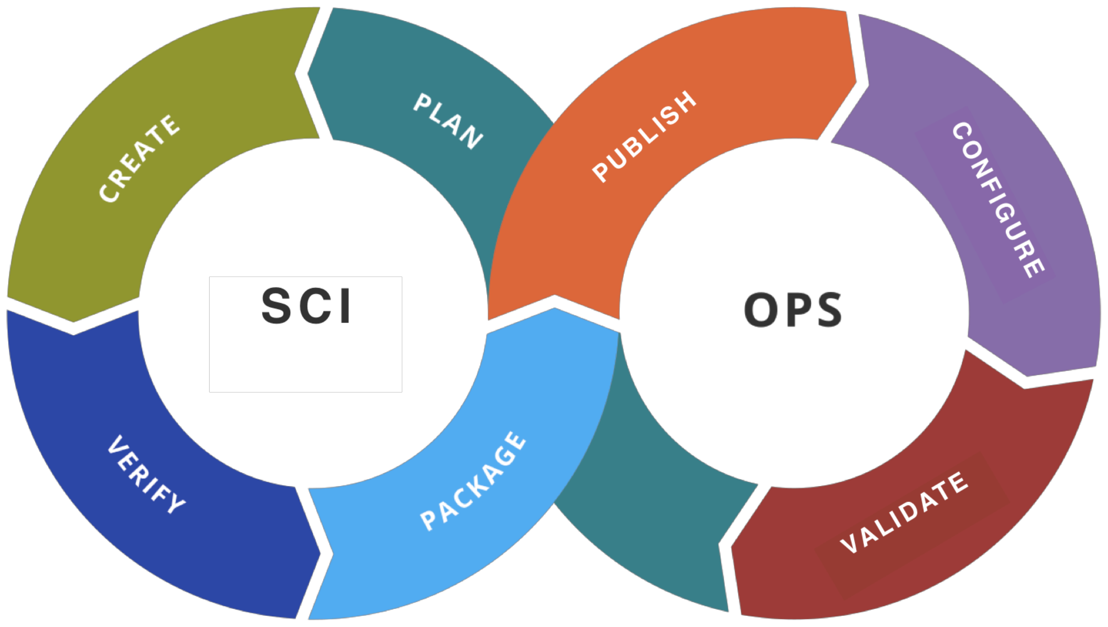


Figure 3.5: The stages of the research delivery process[41]. Research delivery pipelines implement this process by making use of the DevOps toolkit described in Section 3.1.2.

- **Emphasis on validation vs. productivity**. In the software engineering world, when a bug on a system appears is found, fixing the functionality (or performance) of an application is the goal, and this might or not entail finding the root cause of the malfunctioning behavior through systematic means [158,159]. A prominent example of this is how the practices encouraged by the field of validation and verification (V&V) have lower priority in software delivery, where productivity is given a higher priority [160].

- **Long- vs. short-term pipeline maintenance**. In a research curation context, the goal is to maintain an experimentation pipeline that has value for a particular community [48,161]. For example, in the domain of atmospheric sciences, the experimentation pipelines for modeling hurricanes Katrina and Sandy are extensively used as case-studies for testing new modeling techniques, as well as in teaching newer generations of weather prediction experts.

The lack of features in existing DevOps tools with respect to carrying out the activities associated to these three factors is indicative of how software engineering does not value these as much as they are valued in research delivery.

3.2.3 SciOps Principles

Most DevOps principles (Section 3.1.1) can be applied “as is” to the problem of research delivery: infrastructure-as-code, end-to-end automation, self-service, continuous integration and tool-chain homogeneity. I now list the principles that arise in SciOps given the emphasis on the differentiator factors discussed previously. Their collective goal is to ease the validation of scientific claims in research delivery.

3.2.3.1 Validation-as-code

Specify scientific claims in machine-readable format.

In research delivery, instead of deferring to operators (readers, curators, research peers) the task of validating the result of an experimentation pipeline, the original author(s) can instead codify validations with the goal of having self-verifiable pipelines [25]. In this way, ambiguity is avoided and authors can specify exactly what they expect from re-executions of an experimentation pipeline in order to prove their claim. This is an extension to the infrastructure-as-code since the same plain-text formats can be used to specify the expectations on the outcome of a pipeline. This also relates to end-to-end automation since the validation of results can be automated. In Section 4.2, we introduce a domain-specific language (DSL) that can be used to achieve this.

3.2.3.2 Parameters-as-code

Expose parameters of an experimentation pipeline in machine-readable format.

A common experimentation practice is to hard-code parameters in source code or experimentation scripts. Instead, researchers can write a file in plain text format (JSON or YAML) that specify what parameters of an experiment can be modified (e.g. a `parameters.yml` file stored as part of the repository that stores the contents of a pipeline). This allows consumers of research to easily associate parameters to a claim, and to have a clear idea of what can be changed when re-executing and re-using experimentation pipelines.

3.2.3.3 Version-control Infrastructure

Associate IDs to the state of computational infrastructure.

Extend version-control to infrastructure in order to assign hashes to hardware and firmware, so that these IDs can be exposed through a version-control interface. This makes it possible to inform experimenters of changes done at the hardware level, allowing them to be aware of changes that might affect re-executions of a pipeline. This also makes it possible for automation tools to act upon this information. For example, it could be possible to rollback to a previous firmware configuration. This principle can be extended to other cases of dependencies for which versioning techniques are not typically applied (e.g. versioning the content of a database).

3.2.3.4 Good-enough Context Capture

Capture enough contextual information relevant to root cause analysis.

The first question that is asked after an experiment does not produce the expected results is: “what changed?”. This in turn requires to analyze the environments on which the two experimentation pipelines in question (original and re-execution) were executed. Without contextual information for associated to the execution of an experimentation pipeline, it is impossible to answer this question. Which information and how detailed depends on the domain. For example, in systems research, metadata related to the hardware platform (including firmware), as well as the entire software stack, is necessary in order to investigate root causes of irreproducibility. In other cases, domain-specific information is also required, for example in computational science, the seed for the random number generator being used is fundamental in order to determine differences between contexts. The main requirement that makes this principle useful in practice is to represent this information in machine-readable format, so that future automation tools can act upon this data.

3.2.3.5 Continuous Validation

Continuously validate claims.

This is the principle that brings all of the above together. The DevOps principles of end-to-end automation, self-service, combined with the SciOps principles listed above, make it possible to have "push-button" validation, allowing researchers to continuously validate claims. This can potentially be extended to a point where root cause analysis could be automated as well. When contrasting SciOps and DevOps, we can say that this principle replaces the principles of Continuous Deployment and Continuous Feedback, as described in 3.2.2.

3.2.4 The SciOps Toolkit

In the same way that the DevOps principles can be applied in research delivery, most of the tools in the DevOps toolkit can be used "as is" in order to put SciOps principles in practice:

- **Version-control**. Used to store the scripts associated to an experimentation pipeline, as well as to systematically manage software and data dependencies, infrastructure configuration, and any other.
- **Software portability**. Used to ease the burden of re-executing an experimentation pipeline on environments distinct to the one where it originally ran.
- **Dataset management**. Used to manage the input datasets of experimentation pipelines in a way that. Most of dataset management tools provide version-control techniques
- **Continuous integration and deployment**. Used to continuously test the integrity of scripts, so that any changes can be automatically tested and bugs can be identified early.
- **Continuous monitoring**. Used to gather runtime data about an experiment such as performance metrics. This information can then later be used to support a scientific claim or to investigate root causes of irreproducibility.

We identify the need of extending the following:

- **Environment capture**. Existing tools allow to capture relevant information such as versioning of software dependencies. However, they do not capture versioning information at the infrastructure level. In order to address the need to capture infrastructure-level versioning information, existing tools need to be extended with this functionality. Configuration management tools obtain the state of infrastructure by querying at runtime the underlying systems in order to create a list of "facts" about compute and network devices. One way of generating IDs for infrastructure is to find a deterministic way of sorting this metadata, and hash it so that a checksum is generated. This hash ID can then be used as the ID of the piece of infrastructure in question. This information can then be associated to the execution of an experimentation pipeline in order to make this information available so automation tools can act upon them (e.g. notify an experimenter of the change in infrastructure, or rollback to a particular configuration).

- **Infrastructure automation**. Existing tools allow experimenters to specify the state of infrastructure under which a claim is true. However, they do not directly support the ability of rolling back infrastructure. Hardware virtualization software allows to specify the version of the hardware that it is emulating. For certain pieces of infrastructure such as the BIOS or network switches, there are existing efforts adding this capabilities. The ideal is to have holistic rollback features in. In practice, some of this are impossible to achieve, e.g. when storage devices get upgraded, the system cannot be rolled back so that it uses the old hard-drives. However, knowing about this change by having a distinct ID for the infrastructure is a really valuable aid for researchers, e.g. a sanitization phase can run prior to the execution of an experimentation pipeline in order to identify what pieces of infrastructure have changed.

Lastly, we identify new categories that address specific issues in SciOps:

- **Validation automation**. The principle of "Validation-as-code" is not directly supported by existing DevOps tools. Currently, domain-specific tools such as performance regression tools are used in the domain of performance engineering. Data analysis tools such as the ones for Python or R can be used to accomplish this, although they are not designed to do this specifically. Similarly, relational database engines can used to accomplish this, assuming the results of a pipeline are inserted in a database, and the scientific claims are expressed in the form of SQL. In Section 4.2, I introduce a tool for addressing this issue of automated scientific claim validation.

- **Root cause analysis**. Root cause analysis is inherently a domain-specific task. Nevertheless, some subtasks in an root cause investigation deal with domain-agnostic issues such as comparing software dependencies between two executions. The question of "what changed in the latest execution with respect to the original?" can be answered at the level of code by existing software development tools. However, there is no similar tool for applying a "diff" operation between two computational environments. The goal for such a tool would be to take contextual information about "original" and "re-executed" sets of metadata, and produce a result on which things changed, ideally sorted by root cause likeliness (the most likely difference shown first, the second most likely shown next, etc.).

In summary, the goal of the SciOps toolkit is to implement the SciOps principles, mainly by automating the tasks in the research delivery process (Fig. 3.5).

3.3 The Popper Experimentation Protocol

Popper [23,24] is an experimentation protocol and CLI tool for implementing scientific exploration pipelines following SciOps principles. The goal is to aid researchers in bringing agility to the research delivery of their work. The protocol can be summarized in three high-level guidelines:

1. Select one or more DevOps tool for implementing each stage of an experimentation pipeline.

2. Write portable scripts and organize them following the Popper pipeline folder convention (described below).
3. Use a version control system to manage the pipeline scripts, documenting changes in the form of version control commits.

By following these guidelines, researchers can make all artifacts associated to an article publicly available with the goal of maximizing automation when an experiment is re-executed and results are validated.

At the core of the Popper methodology is the concept of a pipeline: a series of Bash scripts that codify a scientific exploration. Following the protocol begins by defining a pipeline for a scientific exploration, i.e. a sequence of high-level steps that are carried out when executing an experiment or analysis. For example, a data analysis pipeline may consist of four stages: (1) obtain a dataset; (2) pre-process the data; (3) run an analysis on the data; and (4) produce plots.

3.3.1 Pipeline Folder Structure

The contents of a folder for a pipeline are shown in Lst. 3.3 (other examples are available on Github[42]). Each stage in a pipeline corresponds to a Bash script, and it codifies what a person would manually type in a terminal otherwise. A stage in a pipeline is a relatively simple list of steps (Lst. **??**), where each step invokes external tools, and passes scripts to them, which are themselves stored in the same repository.

There are four key aspects that make Popper pipelines practical. (1) They are version-controlled, which allows readers to understand what was done over time (and why), mimicking a lab notebook; (2) thanks to virtualization technology at the language-, OS- or hardware-level, pipelines are portable and can be easily re-executed by others; (3) they are automated; and (4) they are self-contained.

[42]https://popper.rtfd.io/en/latest/sections/examples.html

3.3.2 Popper Compliance: Self-verifiable Experimentation Pipelines

While implementing a Popper pipeline can be done in many ways thanks to its toolchain-agnostic approach, in order for a pipeline to be *Popper Compliant*, it must be a *Self-verifiable Experimentation Pipeline* (SEP) [25]. A SEP is a pipeline that carries out the following high-level tasks in every execution:

1. **Code and data checkout**. Code must reside on a version control system (e.g. Github[43], Gitlab[44], etc.). If datasets are used, then they should reside in a dataset management system (Zenodo[45], CKAN[46], Datapackages[47], etc.). The experimentation pipelines must obtain the code/data from these services on every execution.
2. **Setup**. The pipeline should build and deploy the code under test. For example, if a pipeline is using containers or VMs to package their code, the pipeline should build the container/VM images prior to executing them. The goal of this is to verify that all the code and 3rd party dependencies are available at the time a pipeline runs, and that software can be build correctly.
3. **Encode resource allocation**. If a pipeline requires a cluster or custom hardware, resource allocation must be done as part of the execution of the pipeline. This allocation can be static or dynamic. For example, if an experiment runs on custom hardware, the pipeline can statically allocate (i.e. hardcode IP/hostnames) the machines where the code under study runs (e.g. GPU/FPGA nodes). Alternatively, a pipeline can dynamically allocate nodes on CloudLab [150], Grid500K [151], Chameleon [162], or a public cloud provider. These services typically offer infrastructure automation tools such as Geni-lib[48] (Cloudlab), Enos [163] (Chameleon), SLURM [164] (HPC centers), or Terraform[49] (AWS, GCP, etc.). All these tools can be used to automate infrastructure-related tasks.

[43]https://github.com
[44]https://gitlab.com
[45]https://zenodo.org
[46]https://ckan.org/
[47]https://frictionlessdata.io/data-packages/
[48]https://bitbucket.org/barnstorm/geni-lib
[49]https://terraform.io

4. **Explicit parametrization**. Parameters that are relevant to the outcome of the pipeline need to be made explicit, for example by creating a `parameters.yml` file.
5. **Environment capture**. Capture information about the runtime environment. For example, hardware description, OS, system packages (i.e. software installed by system administrators), information about remote services (e.g. version and state of a batch scheduler), etc. Open-source tools such as SOSReport[50] or Facter[51] can aid in aggregating this information.
6. **Results Validation**. Scripts must verify that the output corroborates the claims made on the article. For example, the pipeline might check that the throughput of a system is within an expected confidence interval (e.g. defined with respect to a baseline obtained at runtime), or that a numerical computation is within some expected bounds. This can be done with domain-specific tools [66], or generic data analytics stacks such as Python (Pandas [165]) or R [166].

This compliance criteria ensures that an experimentation pipeline is adhering to the SciOps principles listed in Section 3.2.3.

3.4 Agility in Research Delivery

Popper pipelines can be used to address the problem of communication across the distinct stakeholders of the research lifecycle (Chapter 1.2.1.6). Similarly to DevOps, an outcome of SciOps and Popper is an increase in the rate with which information is communicated throughout the research lifecycle. In this section we describe how SciOps and Popper bring agility to the entire research delivery process.

3.4.1 Creation

At the creation phase, agility is reflected in higher productivity. This is not proper of an academic context but it has been proven empirically in the software delivery context. Concretely, practitioners observe the following:

[50]https://github.com/sosreport/sos
[51]https://github.com/puppetlabs/facter

- Faster personal workflow iteration.
- Minimizing human errors via automation.
- Increased confidence in results.
- Lower cognitive load thanks to systematic management of change.
- Reduced impedance mismatch between scientific workflow and what people do in software delivery.

An aspect that is proper of a research delivery context, is the benefits that come from thinking a-priori of an experimentation pipeline, and its automated validation. Having to go through this exercise has the consequence of experimenters going through experimental design activities and apply best practices in their corresponding domains.

3.4.2 Dissemination

There are three main outcomes of following SciOps and Popper. The first one comes from the "Self-service" principle which results in the removal of dissemination barriers. Using services such as Github or Gitlab, the only impediment for a person to access experimentation pipelines is having the permission to do so. If a repository's visibility is "public", then as soon as a paper is published, the associated pipelines can be accessed by anyone, assuming they can access an internet connection. Secondly, Popper pipelines are self-contained, so there is nothing missing when they get accessed by any stakeholder. Lastly, since the folder structure of a "popperized" repository is well defined, a research dissemination namespace is implicitly defined:

```
<service>/<account>/<repository>/<pipeline>
```

Where:

- **`service`**. Is the name of the version-control service, for example `github`, `gitlab`, `bitbucket`, etc.
- **`account`**. Is the name of an organization or individual account.
- **`repository`**. The name of the repository on the service.

- **`pipeline`**. The name of the pipeline within the repository.

For example, the Popper CLI tool allows researchers and students to search and access pipelines easily.

The Popper CLI is capable of searching searching across multiple services:

```
$ popper search data-science
[####################################] Searching in popperized | 100%


Search results:


> popperized/swc-lesson-pipelines/docker-data-science
```

And obtaining pipelines:

```
$ popper add popperized/swc-lesson-pipelines/docker-data-science
Downloading pipeline docker-data-science as docker-data-science...
Updating popper configuration...
Pipeline docker-data-science has been added successfully.
```

This showcases the agility in dissemination that SciOps and Popper bring to the dissemination phase.

3.4.3 Validation

The validation phase benefits from the "End-to-end automation" principle. Consumers of experimentation pipelines can easily test whether a scientific claim holds on new scenarios.

Another agility aspect is with respect to the investigation of root causes of irreproducibility. When an expectation on the output of a pipeline does not hold, a researcher has the necessary information required to `diff` between original vs repeated execution since the version-control repository records not only inputs but also outputs, as well as relevant environmental information.

3.4.4 Curation

The principles applicable to the validation phase are also applicable to the curation phase. Since curating research entails checking the integrity of an experimentation pipeline, continuous integration and validation help achieve this goal.

In addition, and more importantly, the “Comprehensive Environment Capture” principle is highly relevant since it allows to store information about the context associated to the results of an execution of an experimentation pipeline. This information can be recorded and serve as reference in future curation efforts.

Listing 3.1 An example of infrastructure-as-code. The configuration is written in Terraform's HCL language and requests 3 compute nodes on Google's Cloud Platform.

```
provider "google" {
  region      = "${var.region}"
  project     = "${var.project_name}"
  credentials = "${file("${var.credentials_file_path}")}"
}

resource "google_compute_instance" "www" {
  count = 3

  name         = "tf-www-${count.index}"
  machine_type = "f1-micro"
  zone         = "${var.region_zone}"
  tags         = ["www-node"]

  boot_disk {
    initialize_params {
      image = "ubuntu-os-cloud/ubuntu-1404-trusty-v20160602"
    }
  }

  network_interface {
    network = "default"
  }
}
```

Listing 3.2 Contents of a pipeline.

```
paper-repo/pipelines/gassyfs
|- README.md
|- baseliner
|  |- config.yml
|- docker
|  |- Dockerfile
|- geni
|  |- cloudlab_request.py
|- results
|  |- output.csv
|- run.sh
|- setup.sh
|- validate.sh
```

Listing 3.3 Contents of a ‘setup‘ stage.

```
#!/usr/bin/env bash
set -ex

if [ ! -f "cloudlab/allocation.yml" ]; then
  echo "Error, expecting file cloudlab/allocation.yml"
  exit 1
fi

# [wf] allocate a set of machines on cloudlab
docker run --rm --name=geni-lib \
  -v ${PWD}/cloudlab/allocation.yml:/cloudlab/allocation.yml \
  -v ${PWD}/cloudlab/output:/cloudlab/output \
  -v ${CLOUDLAB_PUBKEY_PATH}:${CLOUDLAB_PUBKEY_PATH} \
  -v ${CLOUDLAB_CERT_PATH}:${CLOUDLAB_CERT_PATH} \
  --entrypoint=python \
  cloudlab/geni-lib:v0.9.7.9
    request

# [wf] sanitize allocation with short-running microbenchmarks
docker run --rm --name=ansible \
  -v ${SSH_KEY_PATH}:/root/.ssh/id_dsa \
  -v ${PWD}/ansible/play.yml:/tmp/play.yml \
  ansible:1.12.0
    ansible-playbook /tmp/play.yml
```

Chapter 4

Reusable Tools For Addressing Research Lifecycle Challenges in Computer Systems Research

In a SciOps context, having tools that aid in the validation of scientific claims has a direct impact on the speed in which experimenters can iterate the research delivery cycle. In this chapter I present a set of tools that aid researchers in the domain of computer systems:

- Section 4.1 presents Fingerprinter, a tool for capturing performance information about a platform in a quick and actionable manner. This tool is used in the creation and validation phases of the research delivery process, allowing experimenters to associate contextual information to the execution of an experimentation pipeline. This information can be used in the analysis of re-executions of a pipeline. Fingerprinter implements the "Good-enough Context Capture" principle of SciOps.
- Section 4.2 presents Aver, a DSL and validation engine. Aver is used in the creation and validation phases of the research delivery process to formally specify and automatically validate scientific claims. Aver implements the "Validation-as-code" principle of SciOps.
- Section 4.3 presents CLAPP, a tool for building performance prediction models

of experiments. These models can be used to identify experimentation pipelines whose associated claims might not hold, when executed on newer platforms. This is helpful in the validation of a large amount of experimentation pipelines since it avoids the need to re-execute them in order to verify a scientific claim, saving computational resources. CLAPP helps to optimize automation of experiment validation, hence helping realize the "Continuous validation" principle of SciOps in systems research.

- Section 4.4 presents *quiho*, a tool for aiding in investigations of root cause analysis of irreproducibility. *quiho* leverages fingerprints to create resource utilization profiles of experiments that allow experimenters to automatically generate hints on changes related to resource utilization. *quiho* also helps to realize the "continuous validation" principle of SciOps.

- Section 4.5 presents CONFIRM, a tool that aids researchers in the creation phase of research delivery. CONFIRM is used prior to the execution of a pipeline in order to determine the number of repetitions that an experiment should be executed in order to obtain results within a user-provided confidence interval. CONFIRM aids experimenters to implement the "parameters-as-code" principle, by capturing explicitly the statistical confidence intervals assumed in an experiment.

4.1 Fingerprinter: Lightweight Profiling of System Performance for Future Reference

Performance profiles of a compute platform[1] are an invaluable aid in performance engineering. Profiling involves recording resource utilization for an application over time. In general, this can be done in two ways: timed- and event-based profiles. Timed-based profiling samples the instruction pointer at regular intervals and generates a function call tree with each node having a percentage of time associated with it, which represents the amount of time that the CPU spends within that piece of code. Event-based profiling

[1]I use the term "platform" to refer to the combination of hardware, firmware and OS. In addition, when we use "application" (or "app") we mean the entire software stack, that is, an application and all its dependencies, excluding the OS.

samples at regular intervals different events at the hardware- and OS-level in order to obtain a distribution of events over time.

Existing profiling alternatives rely on instrumentation of either the application in question or the underlying OS. In either case, the system needs to execute an application in "profiling mode" in order to enable the instrumentation mechanisms that the OS has available for carrying out this task. For example, profiles generated by the popular `perf` Linux tool are obtained for a particular application on a particular platform. Comparisons between profiles can only be done either for the same application across platforms, or for multiple applications on the same platform (but not both). Since one of our goals is to use performance profiles in curation and validation of systems research (see Section 4.1.3), where we need to be agnostic about applications and hardware, these existing approaches are not suitable to our needs.

4.1.1 Application- and Architecture-agnostic Performance Profiling

How can we create performance profile of a platform that is both application- and architecture-agnostic, and that does not require to instrument code nor to run applications in profiling mode? A feasible alternative is to create synthetic microbenchmarks that get as close as possible to exercising all the available features of a system. In this approach, each microbenchmark is only using a specific feature of the platform, for example, only making floating point operations in a loop. Using this battery of microbenchmarks, we can obtain a performance profile of a machine: a performance vector. When this vector is compared against the one corresponding to another machine, we can quantify the difference in performance between the two platforms at a per-microbenchmark level in an architecture-independent way. This performance vector (one dimension per distinct microbenchmark) is the "fingerprint" that characterizes the performance behavior of a platform and has multiple uses, as described in Section 4.1.3. I now describe two alternative batteries of microbenchmarks that I use in my work.

Table 4.1: List of stressors used in this dissertation, along with the categories assigned to them by `stress-ng`. Note that some stressors are part of multiple categories.

stressor	CPU	Cache	Mem	VM
af-alg	X			
atomic	X		X	
bigheap				X
brk	X			
bsearch	X	X	X	
cache		X		
cpu	X			
crypt	X			
full			X	
heapsort	X	X	X	
hsearch	X	X	X	
icache		X		
lockbus		X	X	
longjmp	X			
lsearch	X	X	X	
malloc		X	X	X
matrix	X	X	X	
memcpy			X	
mincore			X	
mmap				X
mremap				X
msync				X
nop	X			
numa	X		X	
oom-pipe			X	
qsort	X	X	X	
remap			X	X
resources			X	
rmap			X	
shm				X
shm-sysv				X
stack			X	X
stackmmap			X	X
str	X	X	X	
stream	X		X	
tsearch	X	X	X	
vecmath	X	X		
vm			X	X
vm-rw			X	X
vm-rw				
vm-splice				X
zero			X	

4.1.1.1 `stress-ng`

`stress-ng` [167] is a tool that is used to "stress test a computer system in various selectable ways. It was designed to exercise various physical subsystems of a computer as well as the various operating system kernel interfaces". There are multiple stressors for CPU, CPU cache, memory, OS, network and filesystem. Since we focus on system performance bandwidth, we execute 42 stressors for CPU, CPU cache, memory and virtual memory stressors (Tbl. 4.1 shows the list of stressors we use). A *stressor* (or microbenchmark) is a function that loops for a fixed amount of time, exercising a particular subcomponent of the system. At the end of its execution, `stress-ng` reports the rate of iterations executed for the specified period of time (referred to as `bogo-ops-per-second`). Every stressor (element in the vector) can be mapped to basic features of the underlying platform. For example, `bigheap` is directly associated to memory bandwidth, `zero` to memory mapping, `qsort` to CPU performance (in particular to sorting data), and so on and so forth.

4.1.1.2 `likwid`

Instead of `stress-ng`, we can obtain performance feature vectors using `likwid-bench` [168], the microbenchmarking tool that is part of the Likwid framework [169]. `likwid-bench` is a benchmarking application together with a framework to enable rapid prototyping of multi-threaded assembly kernels. `likwid-bench` can be thought of a flexible and easily- extensible version of STREAM [170], the popular memory benchmarking tool. With `likwid-bench`, adding a new benchmark amounts to creating a simple text file and recompiling. The framework takes care of threaded execution and pinning, data allocation and placement, time measurement and result presentation. The tool comes with twelve pre-defined kernels to test common CPU and memory operations such as copy, load, and store memory operations, as well as vector multiplications.

The list of available kernels is the following:

- **`copy`**. Standard `memcpy` benchmark. `A[i] = B[i]`.
- **`load`**. Load memory operations.

- `store`. Store memory operations.
- `sum`. Sum of a vector. `A[i] = B[i] + C[i]`
- `stream`. Classical STREAM triad. `A[i] = B[i] + a * C[i]`.
- `triad`. Full vector triad. `A[i] = B[i] + C[i] * D[i]`
- `daxpy`. Linear combination of two vectors. `A[i] = a * B[i] + b * C[i]`.
- `ddot`. Dot product of two vectors.
- `update`. Update operations.

For each of the above, there are five variants:

- **Single-precision**. All the operations listed above are done using double-precision floating point scalars.
- **Multi-threaded**. Multiple streams executing the same kernel. In our case we use as many as cores available
- **Non-temporal memory operations**. By default operations are ordered in such a way that temporal locality is exploited. In this variant, each of the kernels uses non-temporal operations so that they do not observe this optimization.
- **SSE**. Variants of floating point operations that make use of the SSE vector operation instructions of the x86 instruction set.
- **AVX**. Same as above but using the AVX set of instructions.

In addition, there are copy, load and store operations (`clcopy`, `clload` and `clstore`) that are aligned in such a way that the performance of the CPU's last-level cache (LLC) is benchmarked. Thus, the total amount of microbenchmarks that are executed as part of this set is 48.

4.1.2 Obtaining Performance Fingerprints

As part of my work, I created a fingerprinting tool as a reusable component that can be used as a stage in experimentation pipelines. The tool assumes that machines being profiles can run Docker containers or that have a C++ compiler installed. It is available at: https://github.com/ivotron/docker-bench

4.1.3 Using Performance Fingerprints

Performance fingerprints have multiple uses in the research delivery lifecycle in the area of systems research. In this section I describe in detail the use in curation, and give a brief overview of their use in the creation and validation phases since this is described more extensively in Sections 4.3, 4.4 and 4.5.

4.1.3.1 Creation

Scientific claims in systems research are highly dependent on the underlying hardware where experimentation pipeline run. In order to properly control for this source of variability, well-established statistical techniques can be employed, provided there is a large sample of performance measurements for the platforms in question; in other words, a dataset of fingerprints for machines that are used to run experiments can be used as the basis for building statistical tools. If we obtain a sufficiently large amount of fingerprints over a relatively long period of time, we obtain a dataset that can be used as the basis for building statistical tools for aiding researchers. A tool built using this dataset of fingerprints can help answer questions such as “how many times do I need to repeat an experiment so that my claims have high statistical confidence?” or “how representative is a particular machine, with respect to the rest of machines of the same hardware type?”. This is precisely what I have worked on as part of my research [69], which is explained in detail in Section 4.5.

4.1.3.2 Validation

Validating a claim contained in a systems research article entails recreating the environment where the associated experimentation pipeline(s) ran:

1. Hardware. Having access to the same machines where an experiment ran or to the same hardware configuration.
2. Firmware. Using the same firmware versions installed on the hardware.
3. Operating system. Using the same version of the operating system.

4. Configuration. BIOS and OS configuration being the same as in the original study.
5. Software. The entire software stack is the same.

Assuming one can reconstruct all of the above, one should produce similar results with respect to those obtained by the original author(s)[2]. As part of my dissertation work, I studied the suitability of OS-level virtualization (software containers) [171] for addressing point 5 from this list.

When testing an experimentation pipeline on a different hardware platform, that is, when there is a change in any of 1-4, results can be different, depending on the effect that these changes have in the underlying system performance, and on the susceptibility of the experiment to those changes. In a scenario where experimental infrastructure (e.g. CloudLab [150]) hosting thousands of experiments adds new infrastructure to its inventory of machines, testing every pipeline on the new hardware represents a challenge since one would rather skip experiments for which we expect results to hold. In other words, it is desirable to allocate resources to test claims for which we a claim will likely *not* hold, so that we can proceed to explain the root cause, which in turn might result in generating new research insights[^irrepro-philosophy]. My thesis work addresses this by:

- Using fingerprints to create cross-platform performance prediction models that can be used to identify experimentation pipelines whose associated claim(s) will not hold when executed on a new hardware platform (Section 4.3).
- Using fingerprints to create resource utilization profiles that can aid in investigating cases of irreproducibility (Section 4.4).

In these use cases, fingerprints allow to have architecture- and application-agnostic profiling of the underlying platform.

4.1.3.3 Curation

As described in Section 1.2.1.4, one of the main challenges in curation is the preservation of the "performance landscape" at the point in time when a scientific exploration was

[2]This is, assuming that the authors properly controlled for performance variability, as explained in Section 4.5.

carried out. For example, the claims contained in a paper for a distributed storage system are highly dependent on the performance characteristics of the technology that was available at the time the experiments where executed; specifically, the relationship between network, storage, and CPU bandwidth. In an experiment like this, archiving a fingerprint along with the results of an experiment help to contextualize the corresponding performance landscape.

To illustrate the utility of having fingerprints in a curation context, we took the Ceph OSDI '06 paper [172] and reproduce one of its experiments. In particular, we look at the scalability experiment from the data performance section (6.1). The reason for selecting this paper is that we are familiar with these experiments. This makes it easier to reason about contextual information not necessarily available directly from the paper.

The experiments in Section 6.1 of the original paper showed the ability of Ceph to saturate disk evenly among the drives of the cluster. Figures 5-7 from the original paper showed per-OSD performance as the object size varied from 4 KB to 4 MB. Results of the scalability experiment are presented in Section 6.1.3 of the Ceph paper (Figure 8 on the original paper; reprinted below in Fig. 4.1). The goal of this experiment is to show that Ceph scales linearly with the number of storage nodes, up to the point where the network switch is saturated. This linear scalability is our reproducibility evaluation criteria for this specific experiment.

The experiment used 4 MB objects to minimize random I/O noise from the hard drives. We ignore the performance of the `hash` data distribution and increase the number of placement groups to 128 per node, thus we meaningfully compare against the red solid-dotted line in Figure 8 of the Ceph paper.

4.1.3.3.1 Reproducing Results on Similar Hardware

A subset of the hardware used for the Ceph experiments is still available in our laboratory. Each node in the system consist of a 2-core 2212 AMD Opteron 2.0GHz, 8GB of RAM, 1GbE NIC and 250GB Seagate Barracuda ES hard drives. We created a containerized version of the experiment using the 0.87 branch of Ceph. We use docker 1.3.3 and LXC 1.0.6 running on Ubuntu 12.04 hosts (3.13.0-43 x86_64 kernel).

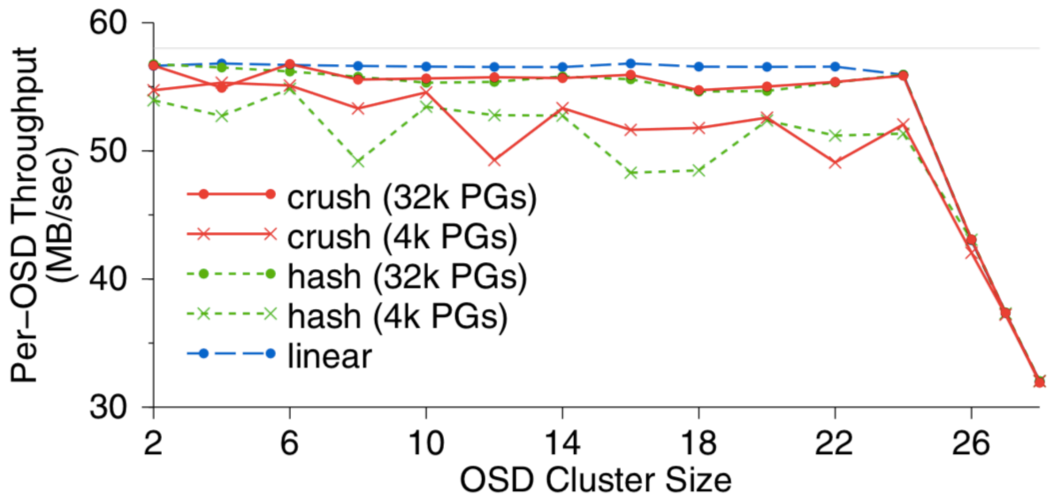


Figure 4.1: Reprinting Figure 8 from the original paper. The original caption reads: "*OSD write performance scales linearly with the size of the OSD cluster until the switch is saturated at 24 OSDs. CRUSH and hash performance improves when more PGs lower variance in OSD utilization.*"

The original scalability experiment ran with 20 clients per node on 20 nodes (400 clients total) and varied the number of OSDs from 2-26 in increments of 2. Every node was connected via 1 GbE link, so the experiment theoretical upper bound was 2GB/s (when there was enough capacity of the OSD cluster to have 20 1Gb connections) or alternatively when the connection limit of the switch was reached. The paper experiments were executed on a Netgear switch. This device has a capacity of approximately 14 GbE in *real* total traffic (from a 20 advertised), corresponding to the 24 * 58 = 1400 MB/s combined throughput shown in the original paper.

We scaled down the experiment by reducing the number of client nodes to 1 (running 16 client threads). This means that our network upper bound is approximately 110 MB/s (the capacity of the 1GbE link from the client to the switch). We throttle I/O at 30 MB/s, so this is our scaling unit (the per-OSD increment). The reason for throttling at 30 MB/s is that, over time, the Seagate disks have aged (they are 10 years old!) and overall performance among the hard drives of our cluster is different from the ~58 MB/s observed in the original paper. In order to amortize, we had to take the lowest common denominator which in this case is 30 MB/s. We throttle I/O by configuring LXC containers with the control group `blkio.throttle.write_bps_device` directive. Fig. 4.2 shows results of this scaled-down, throttled version of the scalability experiment. An open question is to determine if the process of scaling-down and throttling resources can be automated, given the profile repository described in the previous section.

We see that Ceph scales linearly with the number of OSDs, up to the point where we saturate the 1GbE link[3]. We note that we don't see 30 MB/s of net I/O utilization since the current version of Ceph issues two I/O calls on each write request, one to the write-ahead log and another one to the data backend. The original experiments used a prototype version of Ceph that didn't include this atomicity/durability feature. Fig. 4.2 also shows a projection of the original data to our setting. The original result shows better scalability behavior due to newer and more stable hard drives.

[3]The experiment scales linearly up to 4 nodes. At OSD number 5, the 1 GbE link begins to exhibit the effects of network pressure. We empirically corroborated this by re-executing the experiment with two client nodes, in which case the experiment scales linearly up to 8 OSD nodes; at OSD number 9, the two links begin to be pressured (approximately 140 MB/s). This data is available at the repository associated to this use case (see *Section V.C*).

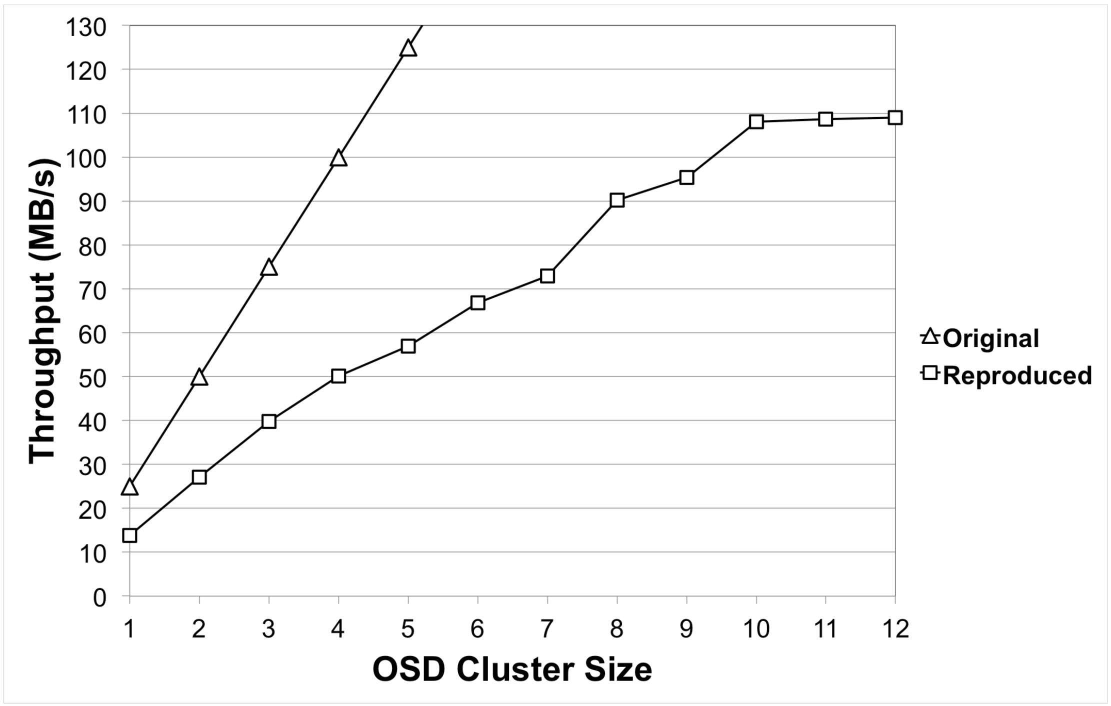


Figure 4.2: Reproducing a scaled-down version of the original OSDI ’06 scalability experiment. The y-axis represents average throughput, as seen by the client. They x-axis corresponds to the size of the cluster (in number of object storage devices (OSD). The square marker corresponds to the average of 10 executions. The line with triangle markers projects the original results to our setting. This projection is obtained by having the 58 MB/s divided by 2 (to reflect the doubled I/O operation of the current Ceph version), i.e. 24 MB/s as the scalability unit of the original experiment.

4.1.3.3.2 Reproducing Results on Different Hardware

So far we have discussed how to reproduce an experiment on the original hardware. But, as we have mentioned before, the challenge is in reproducing experiments on different hardware. Having the experiment implemented in containers allows us to swap components of the underlying hardware and repeat the experiment easily; after all, this is one of the ultimate goals of virtualization, and containers aim at doing it with minimal overhead. Our interest is in measuring the effects that replacing distinct components has on experimental results. Our conjecture is that, for many cases, the mapping methodology defined in the previous section will allow to reproduce results on distinct hardware. As part of our efforts, we are working in characterizing the cases for which our methodology will work and those for which it will not.

Thus, one of our initial goals is to empirically test the repercussions of replacing storage, CPU, memory and network devices (among others). We now present preliminary results on the outcome of swapping distinct storage drives. We re-executed the scalability experiment, swapping four old hard drives with newer models. The results are shown in Fig. 4.3.

The newer hard drives have the capacity to write at ~130 MB/s but we throttle I/O in order to replicate the behavior of our older drives. Standard error markers show that differences for two of the data points are statistically significant. Our expectation was to find complete overlapping points, since at this scalability levels (1-4), variance is relatively low. Additionally, the `blkio` cgroups subsystem has been empirically shown to effectively isolate I/O operations at low loads [173].

After investigating further about the reason of these differences, we found the following. As mentioned earlier, Ceph issues two I/O calls on each write request, one of them being asynchronous. The cgroups `blkio` controller responsible for limiting I/O on block devices (which we configure to 30 MB/s) cannot throttle asynchronous I/O operations since this type of requests go to a queue that is shared at the system level by all containers running in the host. We experientially corroborated this by executing a microbenchmark using FIO that executed the same load (4MB files) on the two hard drives in question, but using direct I/O exclusively. In this case, the performance corresponds to the throttled

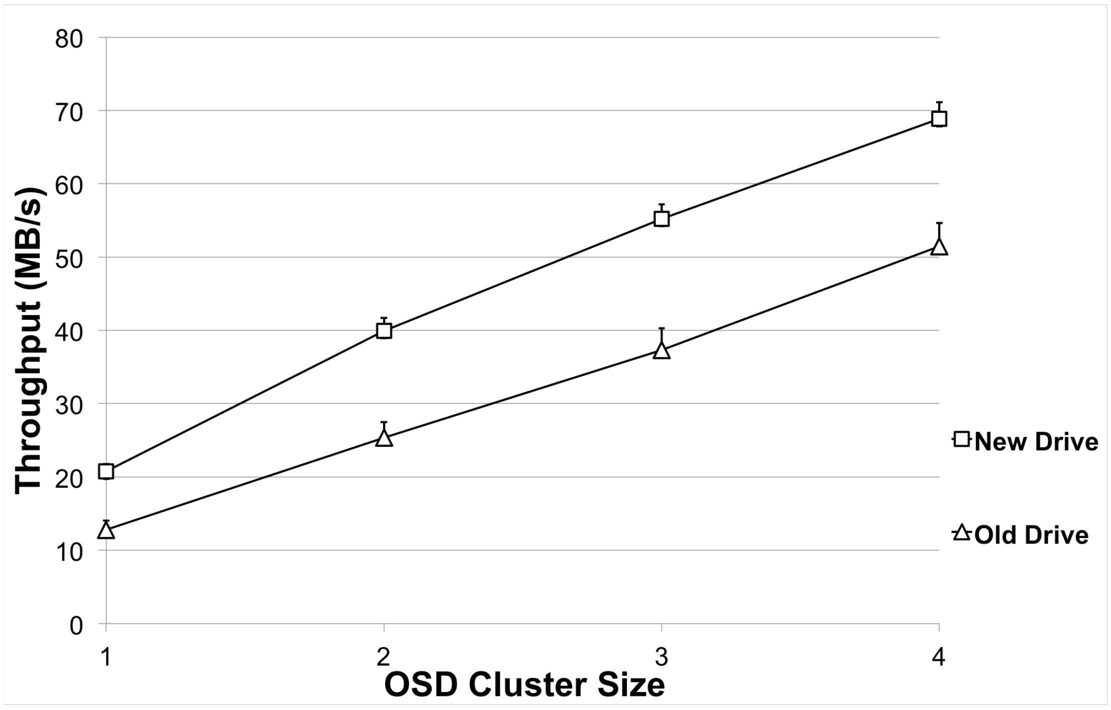


Figure 4.3: Showing the effect of replacing 4 hard drives with newer models. Old hard drives are the same used in the previous figure and correspond to a set of 10 year old 250GB Seagate Barracuda ES (ST3250620NS). New hard drives correspond to 500GB Western Digital Re (WD5003ABYZ) drives. Every data point corresponds to the average (and standard error) of 10 executions.

30 MB/s (lines perfectly overlap). We then executed a mixed workload of both direct and async I/O requests and observed that the newer hard drive performs better than the old one, with similar results as those showed in Fig. 4.3.

4.1.4 Related Work

Performance profilers can be categorized in three broad categories: application-level profiling, OS-level profiling and agnostic. In the first category, applications are explicitly instrumented in order to generate a performance. In the second category, the operating system is instrumented and generates statistics of multiple hardware and software counters available through the Kernel's API. In Linux, the popular Perf Tools [174] toolkit is available. Valgrind [175] is another alternative on Linux that automatically injects applications with profiling directives, although this causes non-negligible effects.

The last category targets the profiling of a platform performance without instrumenting an application and OS code. The Roofline Model [176] is a "visually intuitive performance model used to bound the performance of various numerical methods and operations running on multicore, manycore, or accelerator processor architectures". The Empirical Roofline Toolkit [177] can be used to obtain the characteristics of the underlying platform that are used in the creation of roofline plots. In [178], the authors use the memory access pattern pattern (MAPS) benchmark to characterize the performance of machines.

The Computer History Museum[4] maintains a timeline of storage and CPU technologies, describing the main technological milestones in the history of computers [179]. This information does not include real performance metrics (e.g. bandwidth) of deployed systems; only peak performance of listed components is mentioned.

4.1.5 Future Work

Up to now, we have only looked at characterizing performance of CPU and the volatile memory hierarchy (RAM and CPU cache). Subsequent work can look at characterizing other resources such as network, storage devices and accelerators (e.g. GPUs).

[4]https://www.computerhistory.org

4.2 Aver: Formally Specifying and Automatically Validating Scientific Claims

Currently, scientific claims contained in a paper are expressed in narrative form, which implies that a human needs to interpret the text in order to understand how the result of an experiment backs a scientific claim. In my work, I look at how claims can be formally specified in machine-readable format such that they can be automatically verified. In this section I introduce Aver [66], a simple domain-specific language (DSL) that allows researchers to express scientific claims in the form of expectation statements over tabular data. Aver is also the validation engine that takes these statements and verifies them against a dataset. Aver can be used in two phases of the research lifecycle: creation, where an author expresses the expectations on the results of an experimentation pipeline; and validation, where the statements are automatically verified against a dataset.

This section is organized as follows. I explain the intuition behind Aver in Section 4.2.1. I then present the Aver DSL (Section 4.2.2) and the validation engine (Section 4.2.3). Lastly, I present related (Section 4.2.5) and future work (Section 4.2.6).

4.2.1 From Charts to Formal Scientific Claims

The experimental section of a computer science (CS) article usually includes charts (bar, point plots, box plots, etc.) that are referenced in the text in statements such as "we observe that system A outperforms system B by 30%" or "algorithm A is 10 times faster than algorithm B". The data used to create a chart is usually in tabular form. Thus, if we "reverse-engineer" a chart, we can go from a chart to the data in a table. The main idea behind Aver is this: instead of making a claim based on a chart, one can make it using first-order logic (FOL) statements that reference columns of the corresponding tabular data, similar to how it is done in the relational model [180]. In relational model terminology, a table is a relation and its schema (column names) determines the base FOL predicates that can be used in statements, whereas a row in the table represents a fact. A truth value of a FOL statement is determined by checking a concrete instance of the relation (a table with concrete values). In other words, the contents of a table are checked against the FOL statement in order to assign its truth value.

Table 4.2: Table of runtime measurements for the experiment contained in Fig. 4.1. The header of the table determines the predicates available to create FOL statements. Every row in an instance of this schema (a table with values) is a fact.

`configuration`	`nodes`	`performance`	`net_saturated`
c_1	2	20	False
c_2	4	40	False
...	...	...	...
c_{12}	24	240	False
c_{13}	26	240	True
c_{14}	28	240	True

To exemplify the previous description, take the chart in Fig. 4.1. This chart is accompanied by the following statement: "OSD write performance scales linearly with the size of the OSD cluster until the switch is saturated at 24 OSDs". While we do not have access to the original data, one can assume that it looks like the table presented in Tbl. 4.2. The statement can be expressed in FOL as follows:

Predicates:

- $\mathsf{Configuration}(x)$: x is a configuration of the system (e.g. c_1, c_2, c_3, etc.).
- $\mathsf{Saturated}(x)$: the network is saturated for x (*configuration*).
- $\mathsf{Linear}(x)$: the system performs linearly up to the number of nodes associated to x (*configuration*).

Claim:

$$\forall x : \mathsf{Configuration}(x) \land (\mathsf{Linear}(x) \lor \mathsf{Saturated}(x))$$

In narrative form, the above expresses: "for every system configuration, performance scales linearly with respect to the number of nodes, as long as the network is not saturated". This FOL statement expresses the same claim as the one that appears in the original article. Since this is a "for all" type of statement, in order for it to be true, every row (fact) contained in the table has to result in the statement being true when used to instantiate the predicates, otherwise the claim is false. The particular example in Tbl. 4.2 evaluates this claim to *True*. If the last two facts (rows) on the table had `False` as the value for the `saturated` column, the claim would not hold (the system would not be scaling linearly even though the network would not be saturated).

4.2.2 A First-order Logic-based DSL to Specify Scientific Claims

In order to aid in specifying scientific claims, I defined Aver, a DSL that aims at expressing FOL statements in a succinct, programmer-friendly way. The full BNF grammar can be found at Aver's Github repository[5]. In general, the statements have the form:

```
[when <condition> [and|or <condition>]]
expect <condition> [and|or <condition>]
```

A claim is given via the `expect` clause, which specifies the conditions that must hold in order for the claim to be true. For example, the following expresses the claim "system A shows a 10x speedup over system B":

```
expect
  performance(system='a') > performance(system='b') * 10
```

The `when` optional clause is used to narrow (filter) the facts over which the `expect` conditions are verified. For example, the following expresses the statement "once steady-state is reached, algorithm A outperforms algorithm B in all read-write workloads".

```
when
  steady_state = true and workload = 'read-write'
expect
  performance(algorithm='a') > performance(algorithm='b')
```

For completeness, the example statement given in the previous section can be specified as:

[5]`https://github.com/ivotron/aver`

```
when
 saturated = false
expect
  linear(performance, nodes)
```

Aver has built-in predicates that express a relationship between two metrics that are commonly used in systems research experiments such as `log`, `linear`, `sublinear`, `quadratic`, `strong_scaling`, `weak_scaling`, among others.

As part of my ongoing work (see 4.2.6), I took a sample of 100 articles appearing in top systems research conferences such as VLDB, SC, OSDI, NSDI, SOSP, and ASPLOS. For all of them, the statements made in narrative form can be expressed using FOL. A small subsample of four statements contained in this dataset is shown in Tbl. 4.3. The first column shows the statement as it is written in the article. The second column shows it written in the Aver DSL.

Table 4.3: Example Aver statements corresponding to claims made in papers appearing in systems research articles. Rows 1-4 correspond to articles appearing in VLDB [181], OSDI [182], NSDI [183], and HPDC [184], respectively.

Narrative Form	**Aver**

Hadoop-GIS is more than a factor of two faster than DBMS-X	`expect`
	` time(system='hadoop-gis') <`
	`   2 * time(system='dbms-x')`

in case of failure, both replication and recomputation are faster than restarting the job from scratch	`expect`
	` time(mode='replication') <`
	`   time(mode='restart')`
	`and`
	` time(mode='recompute') <`
	`   time(mode='restart')`

Narrative Form	Aver
In virtually all the communication patterns explored, GlobalFirstFit and SimulatedAnnealing significantly outperform static hashing (ECMP)	`expect` `perf(method='globalfirstfit') >` `perf(method='ecmp')` `and` `perf(mode='simulatedannealing') >` `perf(mode='ecmp')`
the 16nm node shows that the EDP and ED2 of PIM is always better than the host, except for the ED2 of CoMD_EAM3	`when` `config == '16nm'` `expect` `edp(system='PIM') <` `edp(system='host')` `when` `config == '16nm' and` `workload != 'CoMD_EAM3'` `expect` `ed2(system='PIM') <` `ed2(system='host')`

4.2.3 Validation Engine

Aver is also the name of a validation engine that tests statements in the Aver DSL against a dataset. Aver assumes data is in tabular form. A command line interface (CLI) tool accepts one or more statements written in a plain-text file. The statements are then parsed and the resulting abstract syntax tree (AST) is processed in order to map an Aver statement to SQL. The SQL statement results in a value of truth when executed against the given dataset, and this is returned by the engine. The parser and the validation engine can be found at Aver's Github repository.

4.2.4 Automated Validation in Research Delivery

The Aver DSL can be used by researchers to express the claims that their experimentation pipelines are showing to be true. In addition, if they follow an experimentation protocol such as Popper, which results in end-to-end automation of the pipeline, a validation stage can be made part of it, which in turn results in having this verification stage automatically executed. The direct advantage of this is that it removes the need of having the original author(s) or human experts check the output data in order to corroborate the claims made in an article.

Another outcome is the possibility of implementing "continuous validation", which can be seen as an adaptation of DevOps principle of continuous integration (see Section 3.2). In addition, this also allows curators to automatically and continuously check the integrity of an experimentation pipeline.

4.2.5 Related Work

Aver's use of FOL to express statements on tabular data (relations) is inspired on the relational model [180]. Discourse Representation Theory (DRT) [185,186] and File Change Semantics (FCS) [187] are variations of FOL, whose goal is to represent language and meaning using formal logic.

Automated processing of figures contained in an article has also been studied. In general this work can be classified in those techniques that summarize [188] the content of a graph, classifies it [189], or process it in order to extract data from a chart [190–192].

Another category of related work is the one concerned with the automatic extraction of claims from academic articles [193]. Most of this work focuses on biomedical literature [194,195]. Existing work also looks at the problem of representing claims, with most of the work focusing on the use of RDF and semantic web technologies [196,197].

4.2.6 Future Work

One immediate goal for future work is the application of NLP techniques to extract Aver statements from articles appearing in systems research conferences and journals.

4.3 CLAPP: Cross-platform Validation of Scientific Claims

In a scenario where an experimental infrastructure such as CloudLab [150] adds new infrastructure to its inventory of machines, testing every existing pipeline on the new hardware represents a challenge since one would rather skip experiments for which we expect results to hold. In other words, it is desirable to allocate resources to test claims for which it will likely *not* hold, so that we can proceed to explain the root cause, which in turn might result in generating new research insights. In this section I present CLAPP [67], a tool for creating cross-platform learning-assisted performance prediction models with the goal of automating the validation of scientific claims across platforms.

CLAPP is an unobtrusive framework aimed at automatically obtaining performance models for applications on distinct hardware platforms without the need of instrumenting applications or profiling them. The main assumption behind CLAPP is the availability of a multitude of distinct machines when obtaining performance models. When the performance of an application is modeled in CLAPP (Fig. 4.4), the underlying system platforms are baselined with the use of Fingerprinter (Section 4.1). The matrix (or dataset) of performance vectors characterizes the available machines independently from any application and can be used (and re-used) as the basis for applying statistical learning techniques such as statistical regression analysis (SRA). In order to obtain a prediction model, the application under study is executed on the same machines from where the performance vectors were obtained, and a regression model is built and used to predict performance (e.g. runtime). When the performance of an application for a new "unseen" machine is predicted, the machine in question is baselined and the resulting vector is fed to the model, with the resulting value representing the predicted performance.

In this section, I demonstrate that CLAPP can successfully predict the performance of CPU- and memory-bound applications on multiple hardware architectures. We demonstrate it by building models using data obtained from a set of Xeon machines and predicting application performance for AMD, ARM and Atom CPUs with mean absolute percentage error (MAPE) below 5%.

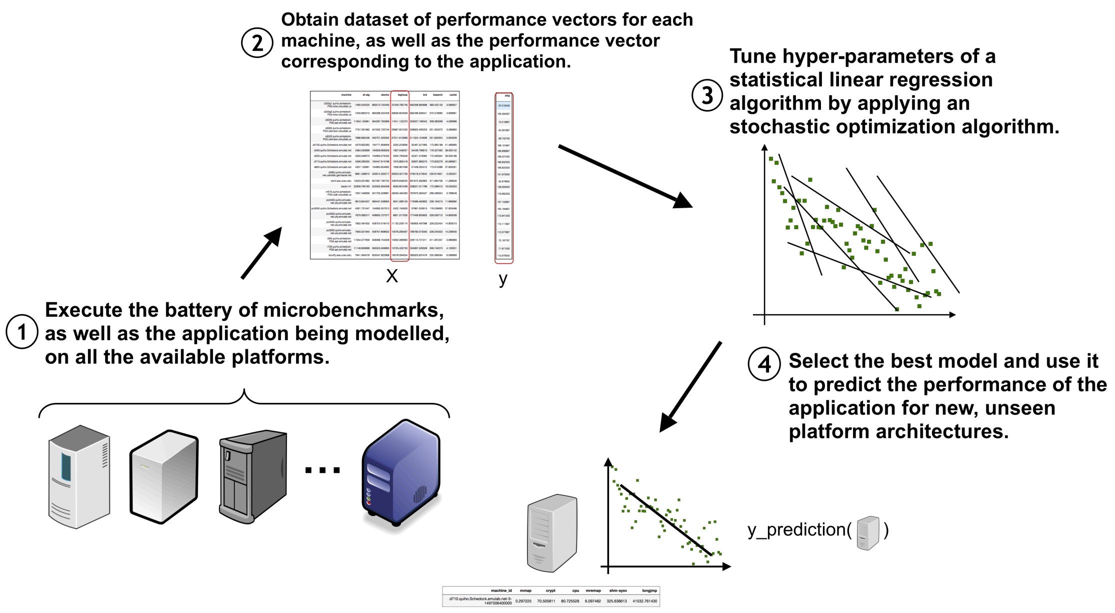

2 Obtain dataset of performance vectors for each machine, as well as the performance vector corresponding to the application.
X
y
3 Tune hyper-parameters of a statistical linear regression algorithm by applying an stochastic optimization algorithm.
1 Execute the battery of microbenchmarks, as well as the application being modelled, on all the available platforms.
4 Select the best model and use it to predict the performance of the application for new, unseen platform architectures.
y_prediction( )

Figure 4.4: CLAPP's workflow for obtaining a cross-platform prediction model for an application.

The next two sections explain the intuition (Section 4.3.1) and technical aspects (Section 4.3.2) behind CLAPP. I evaluate this approach in Section 4.4.3. I close by discussing future (Section 4.3.4) work.

4.3.1 Inferred Performance Modeling

The main goal of CLAPP is to produce prediction models without relying on instrumentation or explicit profiling. In order to achieve this, CLAPP introduces a technique that we term *Inferred Performance Modeling*, where we create a model by inferring the correlation between performance characteristics of an underlying hardware platform and the performance of an application. The main idea is to execute the same application on multiple platforms with distinct performance characteristics in order to infer the correlation between differences at the platform level with differences in application performance. We show an idealized scenario in Fig. 4.5. In this case, a memory-bound application X is executed on three distinct platforms. If we treat the platform and application as black boxes, and only measure runtime metrics, we can use these to infer correlations and create linear regression models. In this idealized example, application performance is perfectly correlated to the underlying memory bandwidth performance of the platform. Using this information, we can create a linear regression model to estimate the performance of the application that are predicated on the underlying memory bandwidth metric.

In order to apply this inferred modeling approach, the performance of a platform needs to be characterized first. While the hardware specification (the "hardware spec") of a machine could be used, the real performance characteristics can only be feasibly obtained by executing programs and capturing runtime metrics. For CLAPP, we run a battery of microbenchmarks to obtain performance fingerprints as explained in Section 4.1.

In Fig. 4.6, we show an example of a matrix of performance feature vectors for a collection of servers (left), and an array of a performance metric for an application on those same machines (right). Every column of a row comes from executing a microbenchmark on that machine. This dataset of microbenchmarks allows us to create a performance prediction model for applications, which we discuss on the next section. Variability patterns of an application (`zlog` in the example) correlate with the same variability pattern of one or more performance microbenchmark(s). Thus, the system subcomponent exercised by the microbenchmark is likely to be also the cause of why the application exhibits such performance behavior. Using this information, we can create a linear regression model to predict the performance of an application on an unseen machine. To do this, the same set of microbenchmarks is used to obtain a performance feature vector for the unseen machine, and using this one can obtain the performance of one or more applications on this unseen machine.

4.3.2 CLAPP pipelines

In this section we describe two ML pipelines that implement the CLAPP approach. These pipelines are evaluated in the next section. In general, for each application being modeled, a pipeline goes through the following steps:

1. Obtain performance fingerprints.

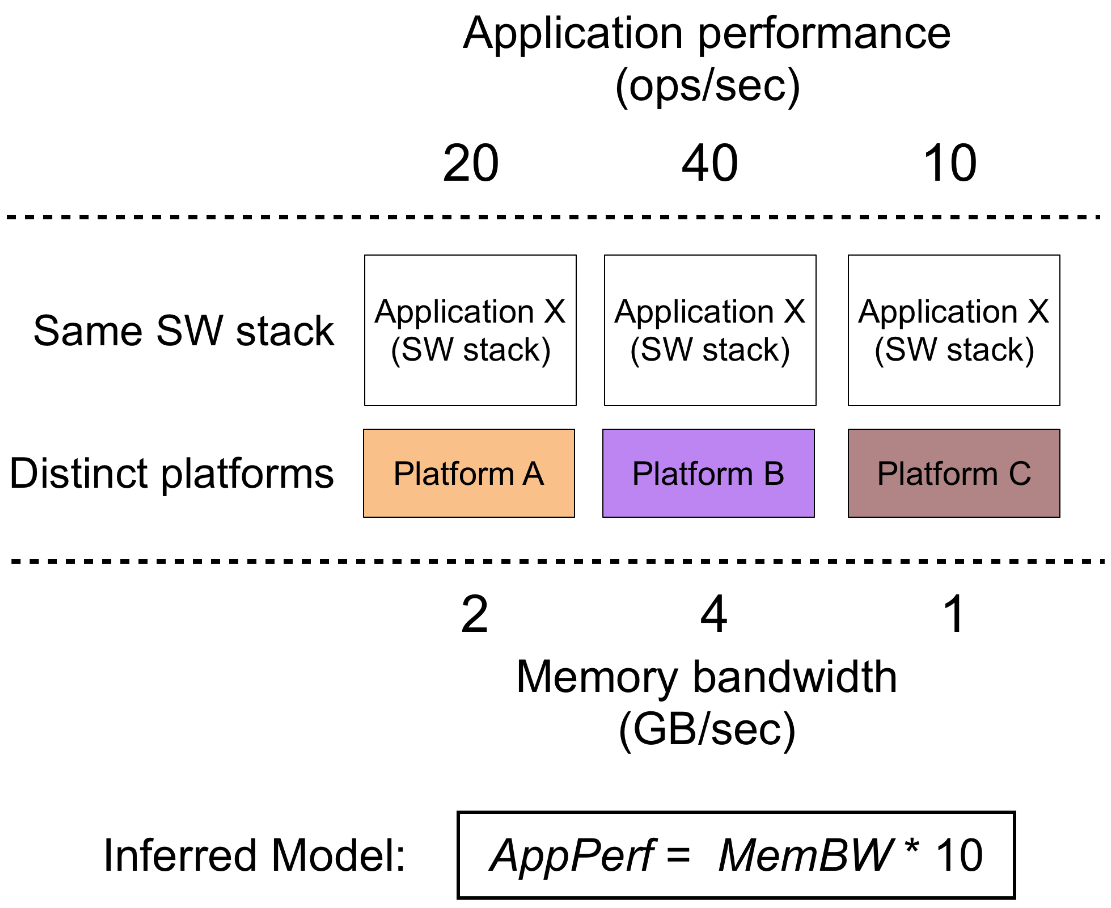
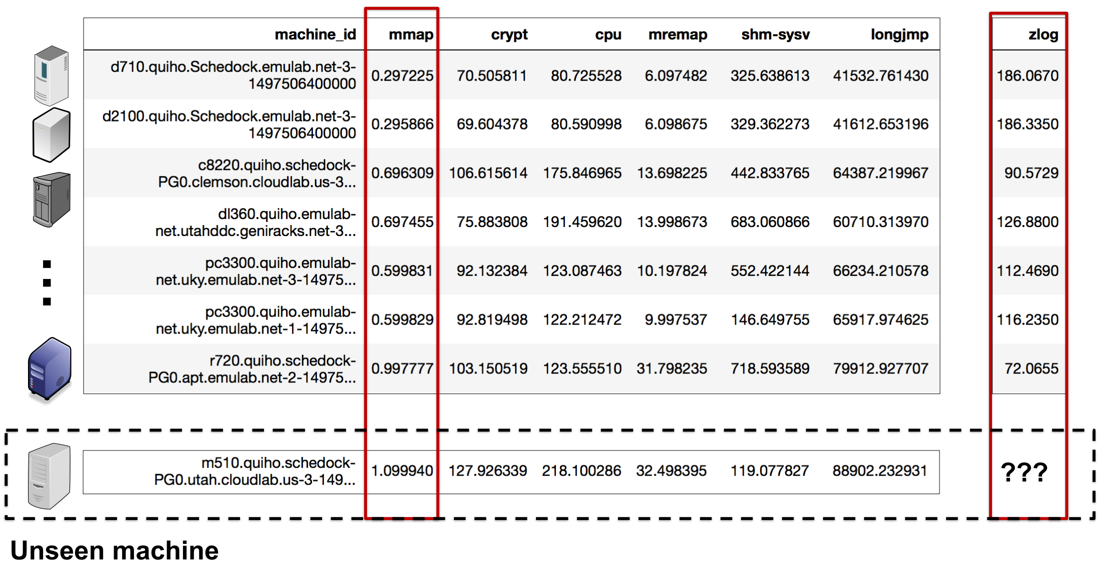


Figure 4.5: Idealized example that illustrates the concept of inferred performance modeling. In this example, we find that the performance of the application in question is directly correlated to the memory bandwidth of the underlying platform. With this information, we can create a linear model to represent the performance of this application, predicated on the underlying memory bandwidth metric.


Unseen machine

machine_id	mmap	crypt	cpu	mremap	shm-sysv	longjmp	zlog
d710.quiho.Schedock.emulab.net-3-1497506400000	0.297225	70.505811	80.725528	6.097482	325.638613	41532.761430	186.0670
d2100.quiho.Schedock.emulab.net-3-1497506400000	0.295866	69.604378	80.590998	6.098675	329.362273	41612.653196	186.3350
c8220.quiho.schedock-PG0.clemson.cloudlab.us-3...	0.696309	106.615614	175.846965	13.698225	442.833765	64387.219967	90.5729
dl360.quiho.emulab-net.utahddc.geniracks.net-3...	0.697455	75.883808	191.459620	13.998673	683.060866	60710.313970	126.8800
pc3300.quiho.emulab-net.uky.emulab.net-3-14975...	0.599831	92.132384	123.087463	10.197824	552.422144	66234.210578	112.4690
pc3300.quiho.emulab-net.uky.emulab.net-1-14975...	0.599829	92.819498	122.212472	9.997537	146.649755	65917.974625	116.2350
r720.quiho.schedock-PG0.apt.emulab.net-2-14975...	0.997777	103.150519	123.555510	31.798235	718.593589	79912.927707	72.0655
m510.quiho.schedock-PG0.utah.cloudlab.us-3-149...	1.099940	127.926339	218.100286	32.498395	119.077827	88902.232931	???

Figure 4.6: Matrix of performance feature vectors over a collection of servers, along with a performance metric for an application. This matrix serves as the basis for building linear regression models for predicting the performance of applications on unseen architectures. To do this, the same set of microbenchmarks is used to obtain a performance feature vector for the unseen machine, and using this one can obtain the performance of one or more applications on this unseen machine.

2. Pre-process data
3. Tune hyper-parameters of the ML model being used.
4. Pick the best parameters and output the corresponding model.

For the first step, we use the two microbenchmarks described in Section 4.1, i.e. Stress-ng and Likwid-bench. The post-processing step is a standard normalization of the data to guard against dimensionality issues. The two ML models that we use in step 3 are the following:

- Gradient Boosting on Decision Trees (GBDT) [198]. In particular, we use the Scikit-Learn interface [199] to the XGBoost implementation of GBDT [200].
- Regression Neural Networks (RNN) [201]. In particular, we use the Keras [202] interface to the Tensorflow library [203].

We optimize parameters using the stochastic algorithm implemented in the Hyperopt Python library [204] (via the Hyperopt-sklearn library [205] for the GBDT pipeline). For each model, before we optimize the hyperparameters of the ML model, we first optimize the main parameter of Hyperopt, which corresponds to the number of trials that the algorithm runs. The higher this value, the more hyper-parameter combinations. The combinatorial space for each model is approximately 200,000. In Fig. 4.7, we show an example of how this is empirically tuned. In this concrete example (and for this application), we find that the ideal is to pick the value of the `maxtrials` parameter to be at least 300.

4.3.3 Evaluation

In this section we evaluate two machine learning (ML) pipelines that implement the CLAPP approach. We first describe the experimental setup (4.3.3.1), followed by the evaluation of each pipeline (4.3.3.2). We then illustrate the trade-off between accuracy and number machines included in the performance fingerprint dataset (4.3.3.3); and briefly discuss runtime performance (4.3.3.4).

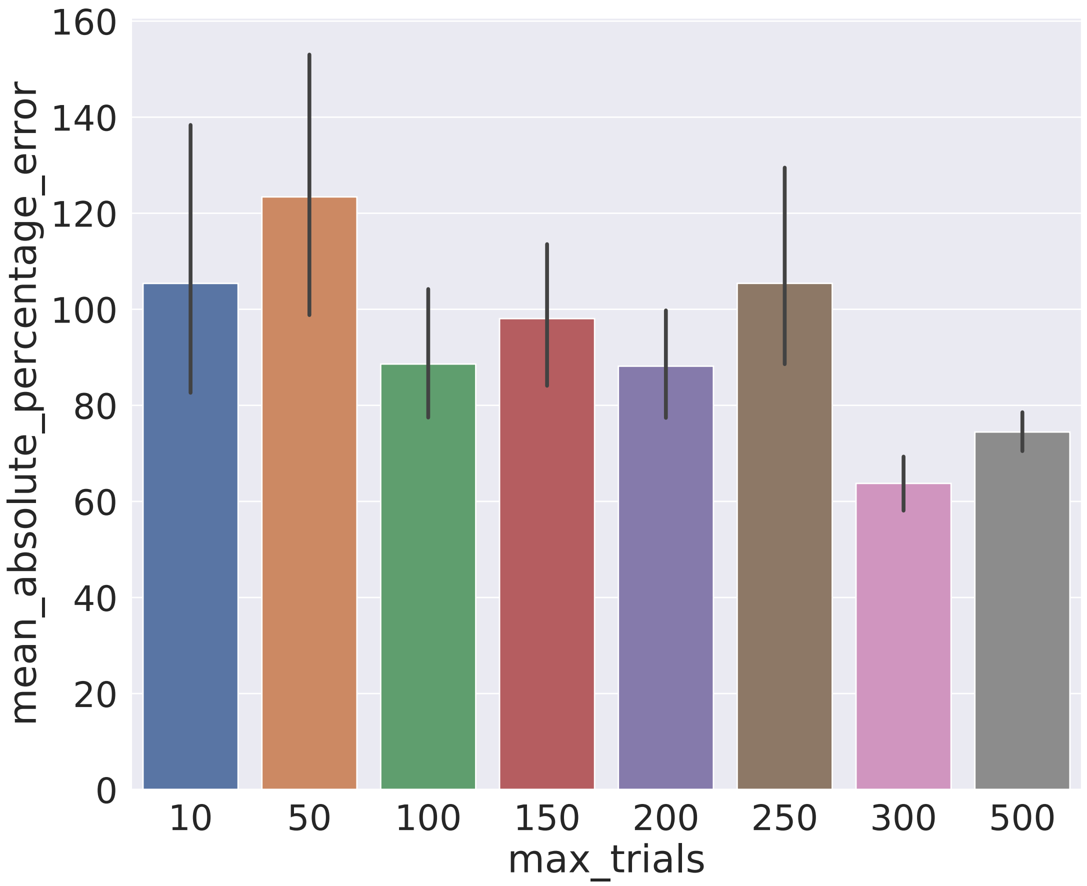


Figure 4.7: Illustration on how the `maxtrials` parameter of `hyperopt` affects MAPE.

Table 4.4: Table of machines used in this study, grouped by architecture.

Label	Architecture	Frequency	Sockets	Cores
pc3000	Xeon 3.00GHz	3.00GHz	1	1
scruffy	Xeon E5-620 v1	2.40GHz	1	4
d710	Xeon E5-530 v1	2.40GHz	1	8
dl360	Xeon E5-2450 v1	2.10GHz	2	8
d820	Xeon E5-4620 v1	2.20GHz	2	8
c8220	Xeon E5-2660 v2	2.20GHz	2	10
d430	Xeon E5-2630 v3	2.40GHz	2	8
c6320	Xeon E5-2683 v3	2.00GHz	2	14
dss7500	Xeon E5-2673 v3	2.30GHz	2	6
xl170	Xeon E5-2640 v4	2.40GHz	1	10
m510	Xeon D-1548	2.00GHz	1	8
c6420	Xeon Gold 6142	2.60GHz	2	16
dwill	Core i5-2400	3.10GHz	1	4
issdm-41	Opteron 2212	2.00GHz	2	2
m400	ARMv8 Atlas/A57	2.4 GHz	1	8
atom	Atom C2550	2.4 GHz	1	4

4.3.3.1 Experimental Setup

Machines used for this study are listed in Tbl. 4.4. All Xeon machines (except `scruffy`) along with the ARMv8 machine are part of CloudLab [150]. The Atom machine is hosted by Packet[6] and the rest are part of UCSC's experimental infrastructure. Horizontal lines group machines by architecture. On each machine, fingerprints are obtained. The applications being modeled consist of Mantevo [206] proxy applications Cloverleaf, MiniAero, MiniAMR, MiniFE, CoMD, as well as HPCCG [207], LULESH [208], SSCA [209], and the PyBench[7] Python suite of tests.

4.3.3.2 Accuracy of CLAPP models

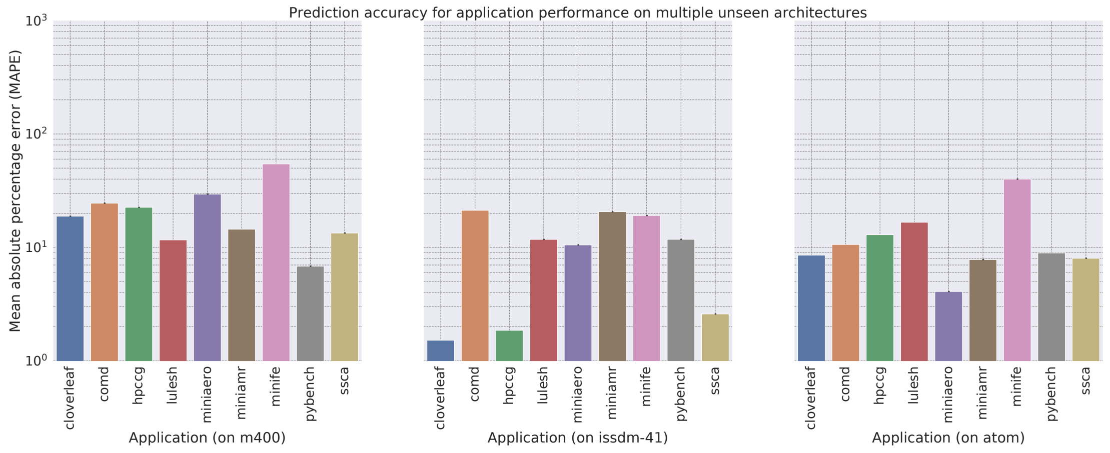


Figure 4.8: Accuracy of prediction models produced by the xgboost pipeline using `stress-ng` performance vectors.

CLAPP's goal is to generate models that can predict application performance across distinct architectures. To evaluate each pipeline, we take all the Xeon machines in Tbl. 4.4 and execute each of the pipeline described in Section **??** to create prediction models for every application mentioned previously. We evaluate the accuracy of a prediction model by first obtaining feature vectors for the three unseen machines with distinct architectures, namely AMD, ARM and Atom machines (labeled `issdm-41`, `m400`

[6]https://packet.net
[7]https://openbenchmarking.org/test/pts/pybench

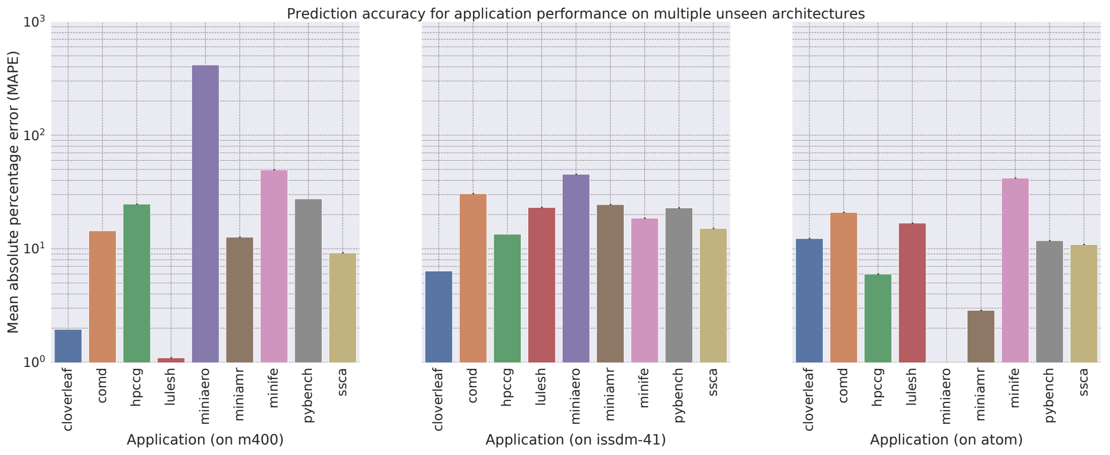


Figure 4.9: Accuracy of prediction models produced by the xgboost pipeline using `likwid` performance vectors.

Table 4.5: Aggregated accuracy results for each pipeline, over all three machines and nine applications.

Pipeline	MAPE Mean	MAPE Standard Deviation
xgboost-likwid	25	47
xgboost-stressng	17	9
keras-likwid	1.9	2.6
keras-stressng	121	158

and `atom`, respectively). Subsequently, we use the fingerprints for these unseen machines and, for each application, we obtain the predicted performance and compare against the actual performance.

Figures 4.8-4.11 show results of each of these. Each figure contains three charts, one for each unseen platform. On each plot, the x-axis denotes the application whose performance is being modeled and the y-axis correspond to the MAPE error metric. In Tbl. 4.5, we report aggregated results for each `[pipeline, dataset]` pair. From this table, we observe that the Keras pipeline using Likwid vectors (Fig. 4.11) generates the most accurate models from the four alternatives.

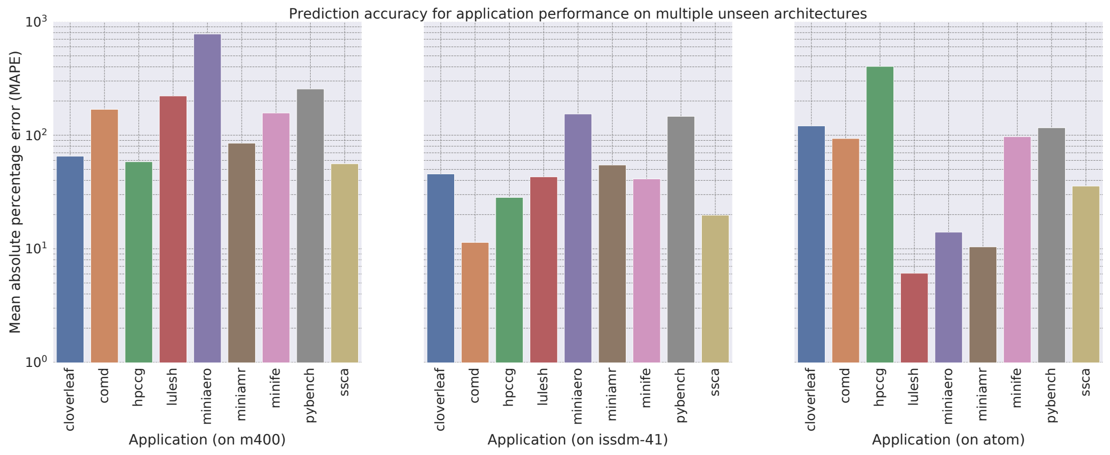


Figure 4.10: Accuracy of prediction models produced by the tensorflow pipeline using `stress-ng` performance vectors.

4.3.3.3 Reducing the Number of Required Machines

The main limitation of CLAPP is the requirement of having to execute the same application in all available machines. In this subsection we answer the question of whether it is possible to reduce the number of required machines by analyzing how the prediction accuracy of CLAPP models is affected by modifying the number of machines from which feature vectors are obtained.

We ran a test where we obtained the powerset of 10 machines, and ran the pipeline, for a single application, for each of the 1023 subsets. We then group models by associating them to the set size where they were obtained from. In other words, we take all models that were created from sets of a single machine, then the group of models from sets with two machines, and so on and so forth.

In Fig. 4.12 we show the accuracy of models grouped by the criteria specified above. The x-axis denotes the size of the machine set, whereas y-axis corresponds to the MAPE as in all preceding charts. As we can observe, the more machines we take into account the better prediction the models make. This can be better appreciated if we look at how the training loss is improved with the number of machines we consider as shown in Fig. 4.13

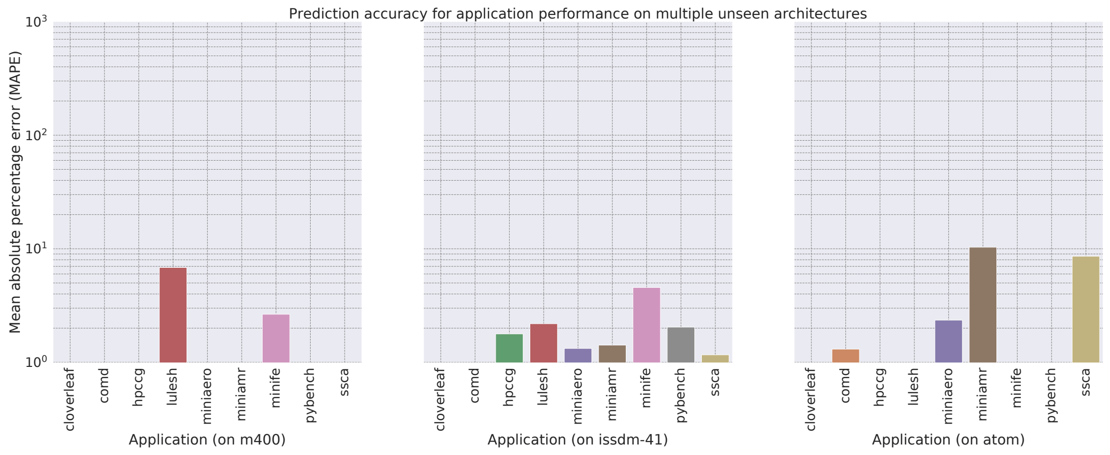


Figure 4.11: Accuracy of prediction models produced by the tensorflow pipeline using `likwid` performance vectors.

4.3.3.4 Runtime Performance of CLAPP

It takes approximately one second to build a CLAPP model using Keras. This is multiplied by the number of iterations executed by the hyperparameter tuning algorithm. Thus, in the worst-case scenario, it takes ~200 seconds (200 iterations of `hyperopt`) to optimize the regression neural network for an application. Since the entire dataset is just 4.5MB (10 data points for each `[machine, app]` pair), this pipeline does not benefit from the parallelism features of the underlying Tensorflow engine.

4.3.4 Future Work

We are currently working in adapting this approach to profile distributed and multi-tiered applications. We also plan to analyze the viability of applying CLAPP in multi-tenant configurations and to profile long-running (multi-stage) applications such as a web-service or big-data applications. In these cases, we would define phases of time and apply CLAPP to each. The main challenge in this scenario is to automatically identify the phases in such a way that we can obtain accurate prediction models.

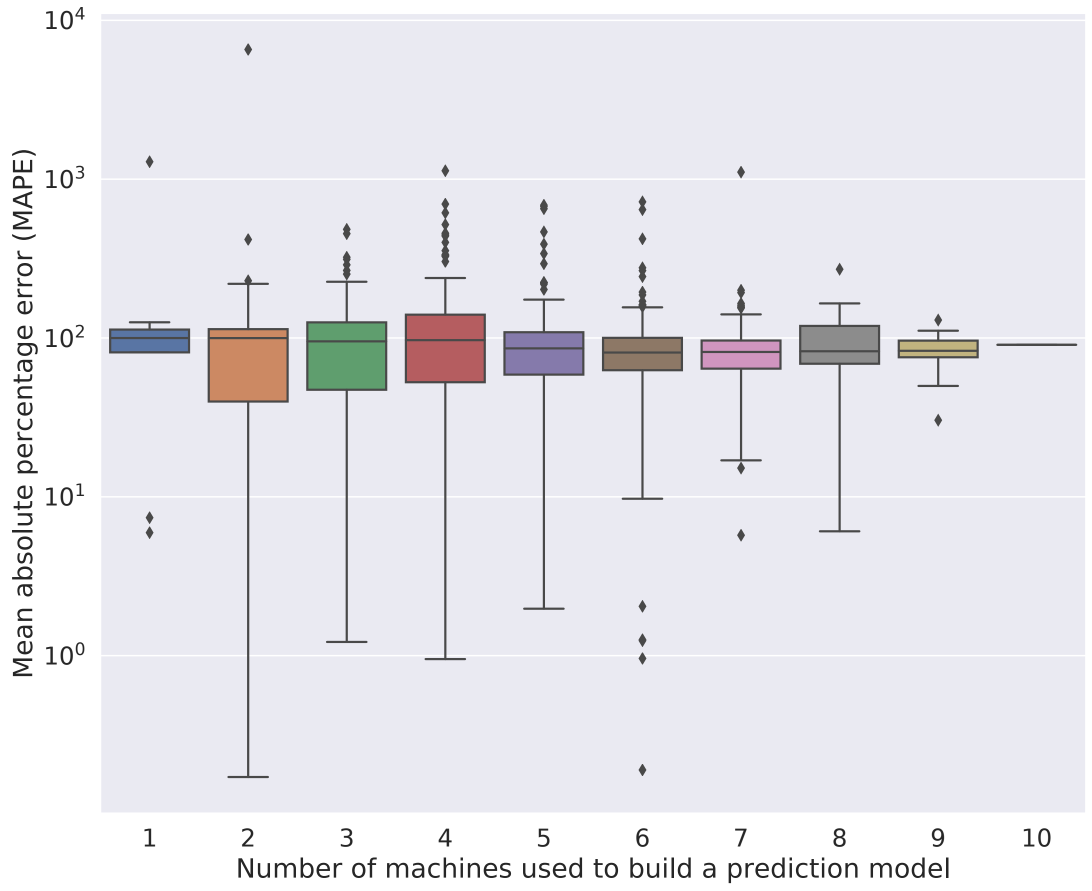


Figure 4.12: Variability of MAPE as the number of machines considered to build a model increases.

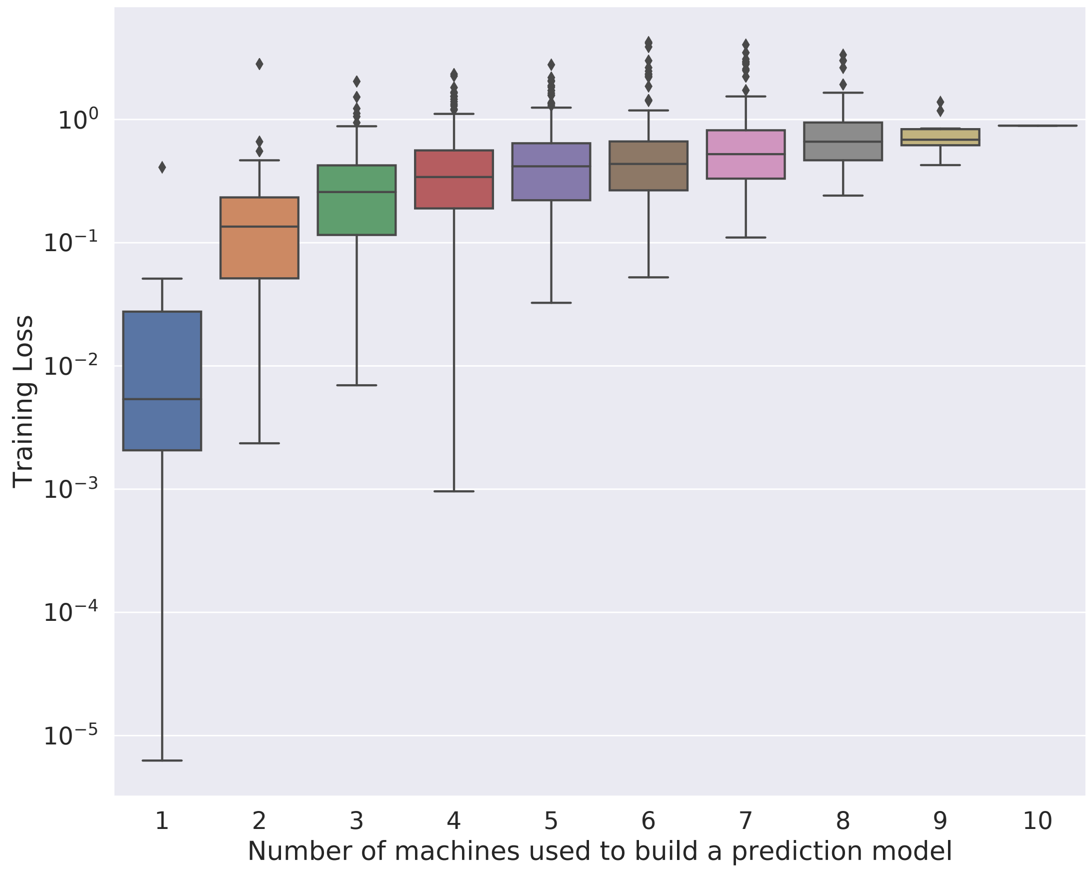


Figure 4.13: Variability of training loss as the number of machines considered to build a model increases.

4.4 quiho: Bottleneck Hints For Root Cause Analysis

When a codified expectation such as the ones expressed with Aver does not hold, it can be considered a performance regression. In this section I present *quiho*, a framework for learning-assisted profiling of resource utilization behavior of an application that can aid in root cause analysis of performance regressions [68]. *quiho* assists the user in identifying the starting points of a root cause investigation.

quiho's approach is aimed at complementing automated performance regression testing by using inferred resource utilization profiles (IRUP) associated to an application. *quiho* is an alternative framework for profiling an application where the utilization of one or more subsystems (e.g. virtual memory) is inferred by applying Statistical Regression Analysis[8] (SRA) on a dataset of application-independent performance vectors. The main assumption behind *quiho* is the availability of multiple machines when exercising performance regression testing, a reasonable requirement that is well-aligned with current software engineering practices (performance regression is carried out on multiple architectures and OSs).

When an application is profiled using *quiho* (Fig. 4.14), the machines available to the performance tests are baselined by executing a battery of microbenchmarks on each. This matrix of performance vectors characterizes the available machines independently from any application and can be used (and re-used) as the foundation for applying statistical learning techniques such as SRA. In order to infer resource utilization, the application under study is executed on the same machines from where the performance vectors where obtained, and SRA is applied. The result of the SRA for an application, in particular feature importance, is used as a proxy to characterize hardware and low-level system utilization behavior. The relative importance of these features constitutes what we refer to as an *inferred resource utilization profile* (IRUP).

In this section, we demonstrate that this approach successfully identifies performance regressions by showing that *quiho* (1) obtains resource utilization profiles for applications that accurately reflect what their code do and (2) effectively uses these profiles to identify

[8]We use the term *Statistical Regression Analysis* (SRA) to differentiate between regression testing in software engineering and regression analysis in statistics.

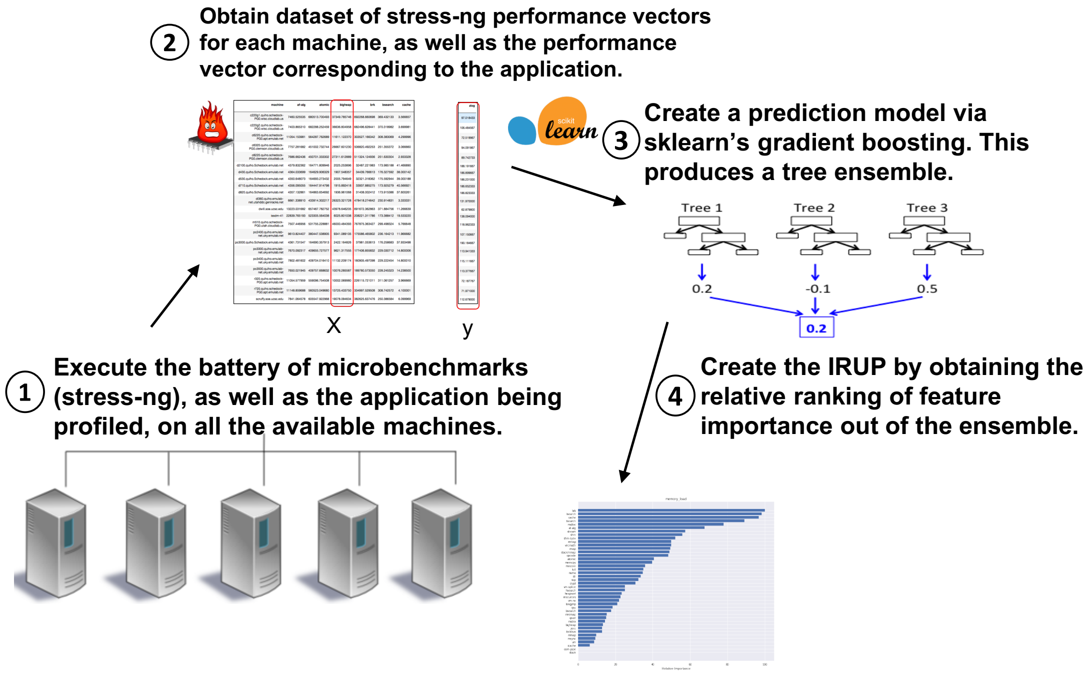

2 Obtain dataset of stress-ng performance vectors for each machine, as well as the performance vector corresponding to the application.
X
y
3 Create a prediction model via sklearn's gradient boosting. This produces a tree ensemble.
Tree 1
Tree 2
Tree 3
0.2
-0.1
0.5
0.2
1 Execute the battery of microbenchmarks (stress-ng), as well as the application being profiled, on all the available machines.
4 Create the IRUP by obtaining the relative ranking of feature importance out of the ensemble.

Figure 4.14: *quiho*'s workflow for generating inferred resource utilization profiles (IRUPs) for an application. An IRUP is used as an alternative for profiling application performance and can complement automated regression testing. For example, after a change in the runtime of an application has been detected across two revisions of the code base, an IRUP can be obtained in order to determine whether this change is significant. IRUPs can also aid in root cause analysis.

induced regressions as well as other regressions found in real-world applications. The contributions in this section are:

- Insight: feature importance in SRA models (trained using application-independent performance vectors) gives us a resource utilization profile (an IRUP) of an application without having to look at the code.
- An automated end-to-end framework (based on the above finding), that aids analysts in identifying significant changes in resource utilization behavior of applications which can also aid in identifying root cause of regressions, and that is resilient to code refactoring.
- Methodology for evaluating automated performance regression. We introduce a set of synthetic benchmarks aimed at evaluating automated regression testing without the need of real bug repositories. These benchmarks take as input parameters that determine their performance behavior, thus simulating different "versions" of an application.

Next section (Section 4.4.1) shows the intuition behind *quiho* and how can be used to automate regression tests. We then do a more in-depth description of *quiho* (Section 4.4.2), followed by our evaluation of this approach (Section 4.4.3). We then survey related work (Section 4.4.4) and close with a brief discussion on challenges and opportunities enabled by *quiho* (Section 4.4.5).

4.4.1 Intuition Behind *quiho*

In the previous section (Section 4.3), we showed how performance fingerprints can serve as the basis for building prediction models by employing inferred performance modeling (Section 4.3.1). Building a prediction model in this way can also serve as a way of profiling resource utilization. If we use performance fingerprints to apply SRA and focus on feature importance [210] of the generated models, they allow us to infer resource utilization patterns by looking at the correlation between microbenchmarks in the fingerprint and the performance of an application. While this can be inferred by obtaining correlation coefficients, proper SRA is needed in order to create prediction models, as well as to obtain a relative rank of feature importances.

Relying on SRA as a way of inferring resource utilization behavior has the practical consequence of *quiho* benefiting heavily from an heterogeneous setup. The more the "performance diversity" of machines that are available for testing, the easier that *quiho* can discover an application's resource utilization behavior. Intuitively, this can be explained as follows. If we run a IO-bound application on distinct machines with very different CPU and memory subsystem performance but similar IO throughput, we won't be able to discover that the application's bottleneck is on the IO subsystem. If we create a more heterogeneous mix of machines, with larger IO performance variability, we can discover that this application is IO-intensive since the performance of the application will vary, depending on the capabilities of the underlying IO subsystem of each distinct machine.

Thus, having high performance variability allows *quiho* to infer resource utilization patterns by discovering the underlying correlations between the performance of microbenchmarks and an application's performance. Since SRA results in creating a performance prediction model for an application, we can rank features by sorting them with respect to their relative performance prediction importance. We call this ranking an *Inferred Resource Utilization Profile* (IRUP), as shown in Fig. 4.15. In the next section we explain how these IRUPs are obtained and how they can be used in automated performance regression tests. Chapter 4.4.3 empirically validates this approach.

4.4.2 System Resource Utilization Via Feature Importance in SRA

SRA is an approach for modeling the relationship between variables, usually corresponding to observed data points [211]. One or more independent variables are used to obtain a *regression function* that explains the values taken by a dependent variable. A common approach is to assume a *linear predictor function* and estimate the unknown parameters of the modeled relationships.

A large number of procedures have been developed for parameter estimation and inference in linear regression. These methods differ in computational simplicity of algorithms,

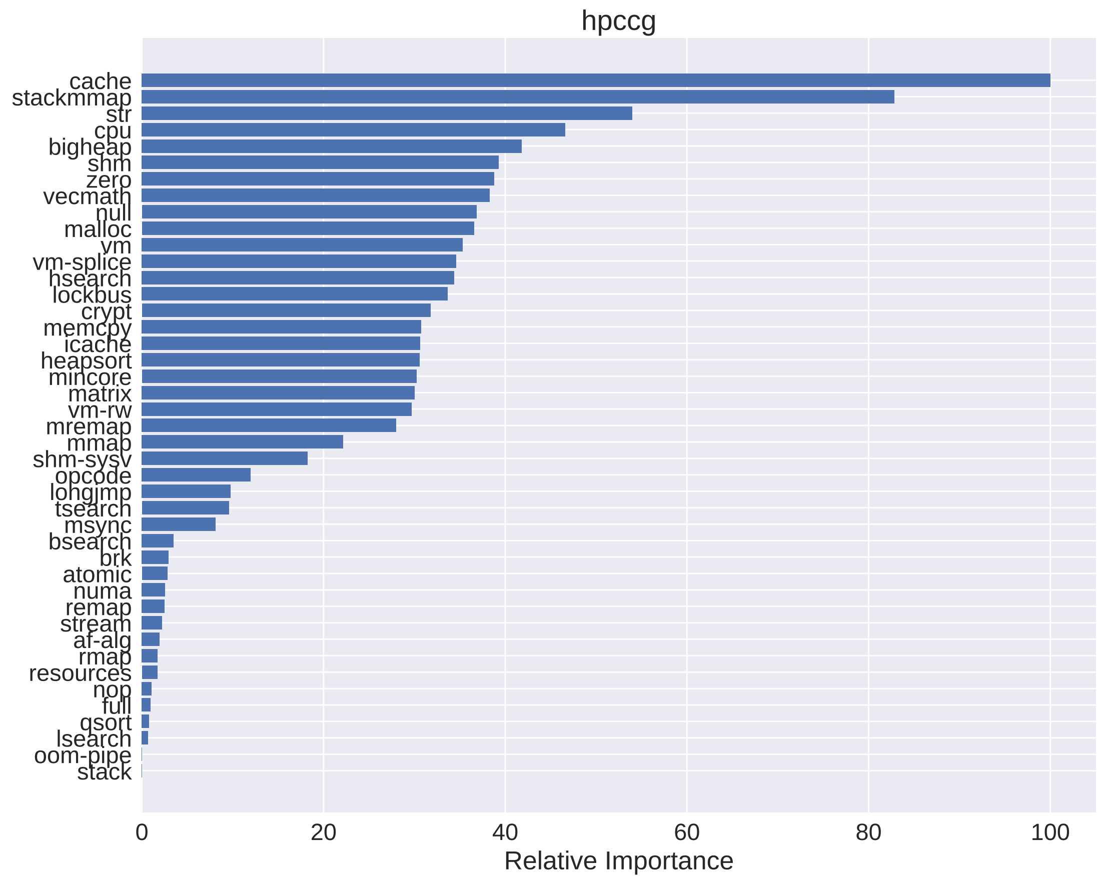


Figure 4.15: An example profile showing the relative importance of features for an execution of the `hpccg` miniapp [207]. The x-axis corresponds to the relative performance value, normalized with respect to the most important feature, which corresponds to the first one on the y-axis (from top to bottom). Chapter **??** describes in detail how feature importances are calculated.

presence of a closed-form solution, robustness with respect to heavy-tailed distributions, and theoretical assumptions needed to validate desirable statistical properties such as consistency and asymptotic efficiency. Some of the more common estimation techniques for linear regression are least-squares, maximum-likelihood estimation, among others.

`scikit-learn` [199] provides many of the previously mentioned techniques for building regression models. Another technique available in `scikit-learn` is gradient boosting [212]. Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees [198]. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function. This function is then optimized over a function space by iteratively choosing a function (weak hypothesis) that points in the negative gradient direction.

Once an ensemble of trees for an application is generated, feature importances are obtained in order to use them as the IRUP for an application. Fig. 4.14 shows the process applied to obtaining IRUPs for an application. `scikit-learn` implements the feature importance calculation algorithm introduced in [213] and is sketched in the following pseudo-code algorithm. Given an ensemble of trees:

1. Initialize an `f_importance` array to hold a score for each feature in the dataset.
2. Take an unseen tree of the ensemble and traverse it using the following steps:
    a. For each node that splits on feature i, compute the error reduction of that node, multiplied by the number of samples that were routed to the node.
    b. Add this quantity to the `f_importance` array (value corresponding to feature i).
    c. Once all nodes are traversed, pick another unseen tree from the ensemble and go to 2.
3. Assign a score of 100 to the most important feature and normalize the rest of elements in the `f_importance` array with respect to this one.

For step 2.a, the error reduction is recursively defined by obtaining the difference between the parent node impurity and the weighted sum of the two child node impurities. The impurity criterion depends on whether the problem is a classification or regression one. Gini or MSE (among many others) can be used for classification. For regression, variance impurity is employed and corresponds to the variance of all data points that are routed through that node.

We note that before generating a regression model, we normalize the data by obtaining the z-score of the dataset. Given that the `bogo-ops-per-second` metric does not quantify work consistently across stressors, we normalize the data in order to prevent some features from dominating in the process of creating the prediction models. In Chapter 4.4.3 we evaluate the effectiveness of IRUPs.

4.4.2.1 Using IRUPs in Automated Regression Tests

As shown in Fig. 4.16 (step 4), when trying to determine whether a performance degradation occurred, IRUPs can be used to compare differences between current and past versions of an application. In order to do so, we apply a simple algorithm. Given two profiles A and B, look at first feature in the ranking (highest in the chart). Then, compare the relative importance value for the feature and importance values for A and B. If relative importance does not have the same value, the importance is considered not equivalent and the algorithm stops. If values are similar, we move to the next, less important factor and the compare again. This is repeated for as many features are present in the dataset.

IRUPs can also be used as a pointer to where to start with an investigation that looks for the root cause of the regression (Fig. 4.16, step 5). For example, if the *stream* stressor (mimics the STREAM benchmark [170]) ends up being the most important feature, then we can start by looking at any code/libraries that make use of this subcomponent of the system. An analyst could also trace an application by capturing performance counters over time and look at corresponding counters to see which code paths make heavy use of the subcomponent in question.

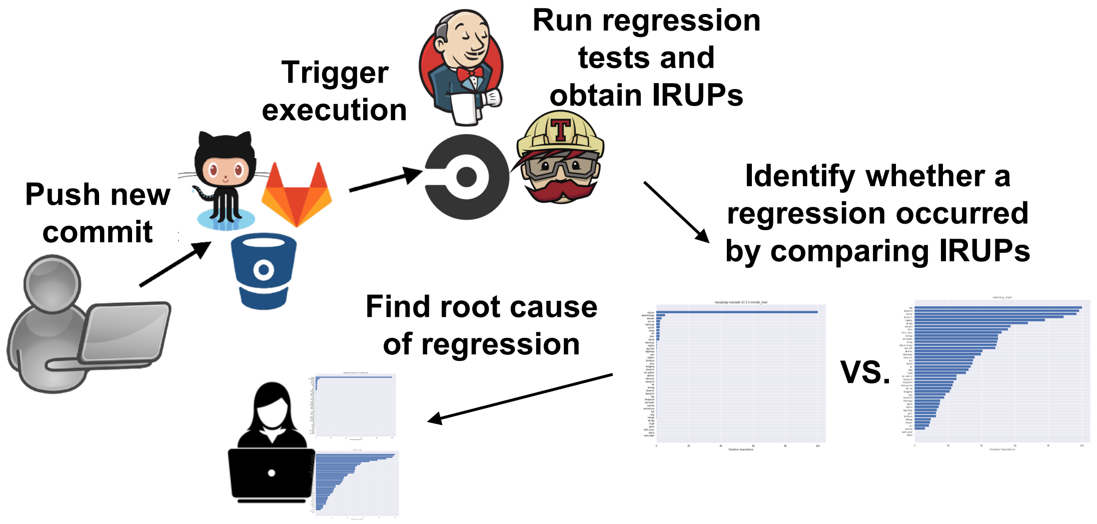

Push new commit
Trigger execution
Run regression tests and obtain IRUPs
Identify whether a regression occurred by comparing IRUPs
VS.
Find root cause of regression

Figure 4.16: Automated regression testing pipeline integrating inferred resource utilization profiles (IRUP). IRUPs are obtained by *quiho* and can be used both, for identifying regressions, and to aid in the quest for finding the root cause of a regression.

4.4.3 Evaluation

In this section we answer the question of how well can IRUPs accurately capture application performance behavior. For more experimental results we refer the reader to [68]. We now demonstrate how IRUPs can effectively describe the fine granularity resource utilization of an application with respect to a set of machines. Our methodology is:

1. Given an application A, discover relevant performance features using the *quiho* framework.
2. Do manual performance analysis of A to corroborate that discovered features are indeed the cause of performance differences.

Fig. 4.15 shows the profile of an execution of the `hpccg` miniapp [207]. This proxy application (or miniapp) [206] is a “conjugate gradient benchmark code for a 3D chimney domain on an arbitrary number of processors [that] generates a 27-point finite difference matrix with a user-prescribed sub-block size on each processor.” [207].

Based on the profile, `stackmmap` and `cache` are the most important features. In order to corroborate if this matches with what the application does, we profiled this execution with `perf`. The stacked profile view shows that ~85% of the time the application is running the function `HPC_sparsemv()`. The code for this function is shown in Lst. 4.1. As the name implies, this snippet implements a sparse vector multiplication function of the form $y = Ax$ where A is a sparse matrix and the x and y vectors are dense. By looking at this code, we see that the innermost loop iterates an array, accumulating the sum of a multiplication. This type of code is a potential candidate for manifesting bottlenecks associated with CPU cache locality [214].

Next, we analyze the IRUPs of other three applications[^brevity]. These applications are Redis [215], Scikit-learn [199], and SSCA [209]. Due to space constraints we omit a similar detailed analysis as the one presented above for `hpccg`. However, resource utilization characteristics of these code bases is well known and we verify IRUPs using this knowledge.

In Fig. 4.17 we show IRUPs for these four applications[9]. The first two on the top correspond to two tests of Redis, a popular open-source in-memory key-value database. These two tests are `SET`, `GET` from the `redis-benchmark` command that test operations that store and retrieve key-value pairs into/from the DB, respectively. The resource utilization profiles suggest that `SET` and `GET` are memory intensive operations (first 3 stressors from each test, as shown in Tbl. 4.1), which is an obvious conclusion.

The next two IRUPs (below) correspond to performance tests for Scikit-learn and SSCA. In the case of Scikit-learn, this test runs a comparison of several classifiers in on a synthetic dataset. Scikit-learn uses NumPy [216] internally, which is known to be memory-bound. The profile is aligned to this known behavior since the `zero` microbenchmark stresses access.

The last application is SSCA, a graph analysis benchmark comprising of a data generator and 4 kernels which operate on the graph. The benchmark is designed to have very little locality, which causes the application to generate a many cache misses. As shown in the profile, the first feature corresponds to the `cache` stressor, which as it was explained

[9]In order to enhance the visualization of the IRUPs we only show the top 5 most important features. Complete profiles can be visualized on the Jupyter notebook contained in the github repository.

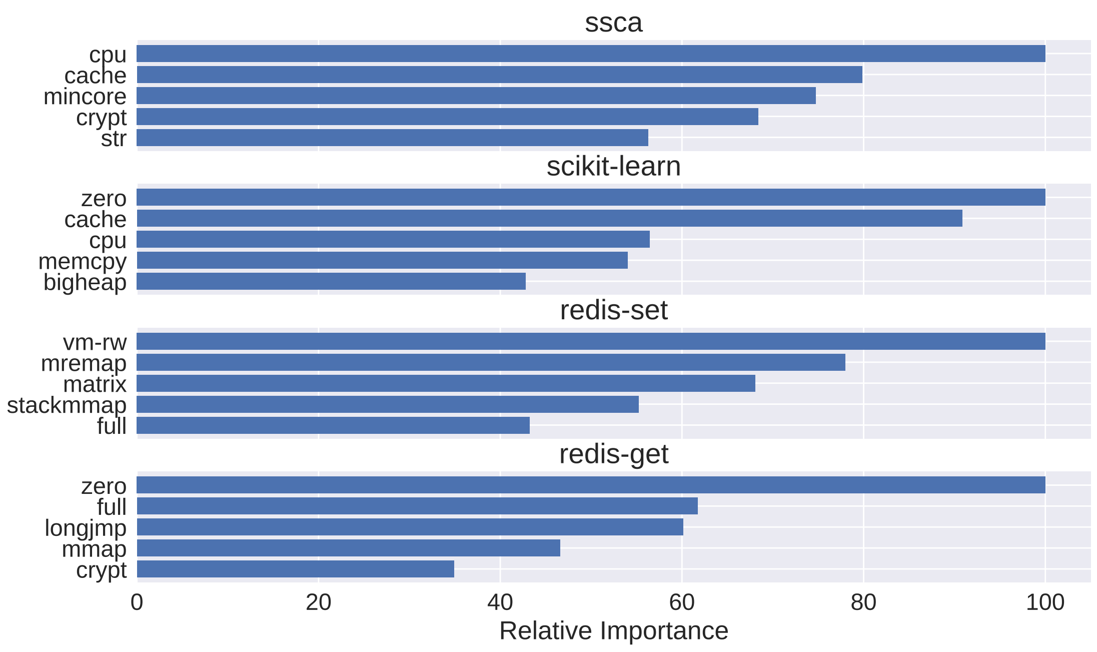


Figure 4.17: IRUPs for the four tests benchmarked in this section. This and subsequent figures show only the top 5 most important features in order to improve visualization of the plots.

earlier, stresses the CPU cache by generating a non-locality workload.

4.4.4 Related Work

Automated Regression Testing. Automated regression testing [217] can be broken down in the following three steps. 1) In the case of large software projects, decide which tests to execute [143]. This line of work is complementary to *quiho*. 2) Once a test executes, decide whether a regression has occurred [218]. This can be broken down in mainly two categories, as explained in [219]: pair-wise comparisons and model assisted. *quiho* fits in the latter category, the main difference being that, as opposed to existing solutions, *quiho* does not rely on having accurate prediction models since its goal is to describe resource utilization (obtain IRUPs). 3) If a regression is observed, automatically find the root cause or aid an analyst to find it [220,221]. While *quiho* does not find the root cause of regressions, it complements the information that an analyst has available to investigate further.

Profiling-based Performance Modeling. Modeling performance based on application profiles has been studied before [178,222,223]. In [178], the MAPS benchmark is used to characterize the performance of machines. These profiles are then convoluted with application traces obtained by the MetaSim tool in order to obtain a prediction on the performance of an application. In [223] the authors use randomized optimization (genetic algorithms) to systematically explore the parameter space of an application in order to create a record of `<input, runtime>` pairs. *quiho* can be used in this case to augment the available information and have an IRUP associated to the inputs of the application under study.

Performance Profile Visualization. An IRUP can be used to visualize performance and thus have a resemblance with a flame graph [224]. In [225] the authors introduce the concept of differential flame grahps, which can be used to visually compare the changes between two or more flame graphs. A similar approach could be applied to IRUPs in order to visualize the differences between two flame graphs.

Inducing Performance Regressions. In [226], the authors analyzed the code repositories of two open source projects in order to device a way of systematically inducing performance regressions. Our methodology instruments an application in order to parametrize performance and control when changes in performance are triggered, as a way of testing methods that are aimed at detecting these changes.

Decision Trees In Performance Engineering. In [227] the authors use decision trees to detect anomalies and predict performance SLO violations. They validate their approach using a TPC-W workload in a multi-tiered setting. In [219], the authors use performance counters to build a regression model aimed at filtering out irrelevant performance counters. In [228], the approach is similar but statistical process control techniques are employed instead. In the case of *quiho*, the goal is to use decision trees as a way of obtaining feature performance, thus, as opposed to what it's proposed in [219], the leaves of the generated decision trees contain actual performance predictions instead of the name of performance counters

Correlation-based Analysis and Supervised Learning. Correlation and supervised learning approaches have been proposed in the context of software testing, mainly for detecting anomalies in application performance [220]. In the former, runtime performance metrics are correlated to application performance using a variety of distinct metrics. In supervised learning, the goal is the same (build prediction models) but using labeled datasets. Decision trees are a form of supervised learning, however, given that *quiho* applies regression rather than classification techniques, it does not rely on labeled datasets. Lastly, *quiho* is not intended to be used as a way of detecting anomalies, although we have not analyzed its potential use in this scenario.

4.4.5 Future Work

Future work can look at adapting the *quiho* approach to profiling distributed and multi-tiered applications. Also one can analyze the viability of applying *quiho* in multi-tenant configurations and to profile long-running (multi-stage) applications such as a web-service or big-data applications. In these cases, one would define windows of time and apply *quiho* to each. The main challenge in this scenario is to automatically define the windows in such a way that we can get accurate profiles.

In the era of cloud computing, even the most basic computer systems are complex multilayered pieces of software, whose performance properties are difficult to comprehend. Having complete understanding of the performance behavior of an application, considering the parameter space (workloads, multi-tenancy, etc.) is challenging. Another application of *quiho* would be to couple it with automated black-box (or even gray-box) testing frameworks to improve the understanding of complex systems.

4.5 CONFIRM: Statistically Sound Experiments on Bare-metal-as-a-Service Infrastructures

In systems research, a gap exists between current experimentation practices and the statistically sound analysis of experimental results that is followed in other domains of empirical research [3]. In this section, I present CONFIRM [69], a tool built in collaboration with colleagues at The University of Utah. CONFIRM can be used in bare-metal-as-a-service infrastructures by administrators and researchers to determine how many times to execute an experiment, given a target confidence interval, in order to obtain results that lay within this interval.

4.5.1 Variability in Systems Experiments

The fundamental way variability impacts systems research is that it affects our *confidence* in the statistical power of our results and the correctness of conclusions that we draw. When we run experiments and calculate statistics (mean, median, etc.) we are producing *empirical* statistics from a sample (a finite number) of a notional *population* (an infinite number) of executions. As we run more *repetitions* of an experiment, we can be more confident that our empirical distributions are close to the population distributions, and for key statistics such as the mean and median, we can compute *confidence intervals* (CIs).

For a chosen *confidence level* α, a CI defines a range in which we are α% sure that the population mean lies. For example, a sample mean of 10.0, with a CI of $9.9 - 10.1$ at 95% confidence indicates a 95% confidence that the true mean lies within $r = 1\%$ of our estimate 10.0. In order to make a strong statement that one sample mean is higher than another, their CIs should not overlap [229]; if they do, it is possible that the true population means have the reverse relationship. When an experiment is analyzing a small effect (for example, a 5% performance improvement), a wide CI may invalidate the conclusion.

Statistical methods fit into two broad classes: parametric and nonparametric techniques. The former class, which is more well-known, relies on the assumption that the analyzed data stems from known probability distributions, typically the Normal/Gaussian distribution. A variety of closed-form expressions for statistics of interest enable powerful parametric analysis. In contrast, nonparametric techniques are used when the probability distributions are *unknown*, and require fewer assumptions. Many studies suggest that the normality assumption does not hold for the data obtained in computer systems experiments, especially when the data includes measurements of performance. This applies both on a single machine [230] and in parallel programs running on supercomputers [231]. Indeed, most of the data in our dataset (Section **??**) does not follow the normal distribution. Thus, we adopt nonparametric statistics for the remainder of this section, and recommend that, for performance experiments, these methods be used unless normality can be demonstrated. In [3] and [232], the authors provide advice for statistically sound performance analysis and argue for applying robust nonparametric techniques.

A natural question is how many repetitions of an experiment are likely to be needed to achieve a sufficiently narrow CI (e.g., indicating that the empirical median differs from the true median by no more than $r = 1\%$) for a given confidence level α (e.g. 95%): we want to be sure to run enough repetitions to be confident in our results, but don't want to waste time running more than necessary. We use $E(r, \alpha, X)$ to represent this value for a set of experiment results X. Finding $\check{E}(X)$ for parametric models is straightforward, as most such models have a closed-form equation that uses an estimate of the variance of X, obtained by running a handful of exploratory experiments. In the nonparametric case, this number is harder to find, since we cannot make any assumptions about the distribution and there is therefore no equation we can use. In Section 4.5.3, the CONFIRM tool is introduced. CONFIRM makes it easy for experimenters to get these estimates, and it is based on a resampling technique for estimating $\check{E}(X)$ for nonparametric models.

4.5.2 A Dataset For Building Statistical Tools

In order to build statistical tools for experimenters, we first need to create a dataset from where to obtain the statistical descriptors. Over a period of 10 months, from May 20, 2017 to April 1, 2018, we collected performance fingerprints on servers that are part of the three CloudLab [150] clusters. The microbenchmarks were run while servers were *not* allocated to other users, meaning that they did not affect, nor were they affected by, other users of the facility. The dataset is open and publicly available [233].

The fingerprints were gathered from CloudLab's three primary clusters: Utah, Wisconsin, and Clemson. Servers at each site are divided into a small number of distinct homogeneous *types*; no sites currently have overlapping types. All servers we tested are interconnected via a 10Gbps "experiment" network within each site. At the time of our tests, each of these sites had two "dominant" types consisting of tens to hundreds of servers. Some sites have types with only a few instances containing specialized hardware such as GPUs or many disks; we did not test these types to avoid consuming CloudLab's scarcest resources.

From the period of May 20th 2017 to April 1st 2018, we collected $10,400$ total runs from 835 total machines. Since each run involved execution of a multitude of benchmarks in different configurations, we ended up with a total of $892,964$ distinct data points over this period. We use the term "configuration" to refer to the combination of hardware type, configuration, and benchmark settings. For example, the possible memory configurations come from varying hardware type, socket number, single- or multi-threaded operation, frequency scaling, and type of memory operation; this results in 590 possible configurations for memory. Similarly, there are 96 possible configurations for storage, and 27 possible configurations for network tests. Each data point in the dataset comes from executing one configuration.

4.5.3 CONFIRM: A Tool for Designing Statistically Sound Experiments

Given that some amount of variability is inevitable, we turn to a perennial question for experimenters: "How many repetitions do I need to run in order to be confident in my results?" As described in Section 4.5.1, given a set of measurements and a desired confidence level (such as 95%), we can compute a confidence interval (CI) for the mean or median. A standard procedure is to "invert" this calculation, and for a given desired confidence level and CI width, estimate how many repetitions are likely necessary to achieve the desired confidence.

When assuming normality, there is a closed-form equation to calculate this estimate [229]; the main input to this equation is an estimate of variance, typically obtained by running a small number of trial runs. In the nonparametric space, there is no closed-form equation, so producing such an estimate requires a more complex technique. We have developed such a technique using *resampling*:

For a set of collected measurements X with n values, we randomly select a subset of $s \leq n$ values for which we estimate the bounds of the CI for the median as described in Section 4.5.1. We shuffle X, select another subset of s values, and obtain new estimates of the CI. After we repeat this process c times, we calculate the means of the lower and upper CI bounds. Obtained using *sampling without replacement*, each of these random selections or "trials" represents a hypothetical scenario where a smaller, partial subset of measurements was collected by an experimenter. The aforementioned averaging eliminates the dependence of the results on the properties of a particular subset and provides an aggregate view on the convergence of the CI observed across many trials. The results presented in the rest of this section are obtained using $c = 200$. To estimate the recommended number of measurements $\check{E}(X)$, we start at $s = 10$, assuming that smaller subsets are insufficient to estimate nonparametric CIs reliably and should not be considered. Then, we increase s until $s = n$ or the mean CIs fit within the desired error bounds. In the former case, we conclude that these n samples are insufficient for meeting the stopping condition, while in the latter case, we note that the experimentation could have stopped after $\check{E}(X) = s$ measurements according to the selected allowed error and confidence level.

We have implemented this technique in a service we call CONFIRM, for CONFIdence-based Repetition Meter. This dashboard imports our benchmarking datasets and facilities interactive nonparametric analysis of CIs for measurements collected from individual servers, groups of servers, and entire hardware types available on CloudLab. The tools is available at https://confirm.fyi.

4.5.4 Related Work

In [234], the authors present a profiling study of a Warehouse-Scale Computer where they analyze 12 to 36 months worth of performance counter metrics for applications running on Google data centers. The study focuses on microarchitecture-level statistics to identify hotspots in distributed applications, main memory and CPU cache latencies, among others. In contrast, we focus on coarser-grained metrics such as runtime and bandwidth of microbenchmarks with the goal of taking into account the points of view of both system administrators and users. Similar studies have focused on other cloud platforms such as Microsoft's Azure [235]. Other related profiling efforts have the goal of improving the scheduling of applications on shared infrastructure by identifying and reducing contention between applications [236,237]. More recently, in [238], the authors present a study of the impact of slow failures (i.e. "hardware that is still running and functional but in a degraded mode, slower than its expected performance") found in large-scale cluster deployments in 12 institutions.

In [239] the authors describe a suite of tests composed of of microbenchmarks that run continuously over the entire Grid5000 infrastructure. The heuristic to decide which tests to run and where is similar to ours, but in our case we prioritize testbed coverage. In [240] a set of open questions for experimental testbeds are outlined, with respect to reproducibility of experiments. In particular, the topic of "Respective Responsibilities of Testbeds and Experimenters"" poses the questions of "How far should testbeds go with providing advanced services to experimenters? What should be left as a burden for experimenters?" As part of our work, we have introduced the foundation for a new service that aids experimenters in getting a better understanding of the variability of the underlying platform with respect to the performance of basic subcomponents (CPU, memory bandwidth, network and storage).

4.5.5 Future Work

In this study, we have deliberately focused on the set of hardware resources whose performance is of the most interest in the CloudLab testbed. Differences due to system software and libraries—kernels, compilers, memory allocators, etc. should not be discounted, and there are many more hardware metrics that are of interest. We hope to expand our study to include these factors in the future.

Listing 4.1 Source code for bottleneck function in HPCCG.

```
int HPC_sparsemv(HPC_Sparse_Matrix *A,
                 const double * const x,
                 double * const y)
{

  const int nrow = (const int) A->local_nrow;

  for (int i=0; i< nrow; i++) {
    double sum = 0.0;
    const double * const cur_vals =
     (const double * const) A->ptr_to_vals_in_row[i];

    const int    * const cur_inds =
     (const int    * const) A->ptr_to_inds_in_row[i];

    const int cur_nnz = (const int) A->nnz_in_row[i];

    for (int j=0; j< cur_nnz; j++)
        sum += cur_vals[j]*x[cur_inds[j]];
    y[i] = sum;
  }

  return(0);
}
```

Chapter 5

Black Swan: Ongoing and Future Work

The goal of Popper (Chapter 3) is to bring the same methods and tools used for DevOps to scientists and industry researchers. Our experience in the past four years developing and evangelizing the use of Popper in multiple scientific domains has allowed us to identify opportunities where open-source software (OSS) can be used to close the existing gap in current research practices. While the Popper protocol is relatively easy to follow (wrap experimentation pipelines in the form of a sequence of Bash scripts), for many practitioners it still represents a big leap between current practices and where OSS development communities are today (automated, portable and versioned software testing pipelines). To this end, in this chapter I present our ongoing and future work: *Black Swan*[1], a platform for *practical* reproducible research. In a nutshell, Black Swan enables the agile delivery of research generated in universities and other research institutions, significantly accelerating technology transfers between research and operational environments.

This chapter is organized as follows. Section 5.1 expands on the need for Black Swan, while Section 5.2 presents the components of the proposed platform and how they address the current gaps. Section 5.3 illustrates the utility of the platform by describing use

[1]Black swans are typically used to illustrate the concept of falsifiability: the statement "all swans are white" is made falsifiable by defining the condition under which it would be false, i.e. finding one or more non-white swans. In this case, the statement was actually proven false by the discovery of black swans in Australia in the 1600's. We envision *Black Swan* (the platform) to be a tool for researchers to *easily* find black swans in their computational or data science theories (i.e. identify when their claims are false) and, more importantly, allow them to investigate **why**.

cases where Black Swan would be used. We close by discussing challenges in Section 5.4.

5.1 Popper In Practice

Since a Popper pipeline (Section 3.3) is, fundamentally, a list of Bash scripts (with a specific execution order) stored in a version control repository with a pre-defined folder structure, implementing a self-verifiable pipeline (SEP) [25] (Section 3.3.2) is an arguably straightforward habit to adopt.[2] However, based on our first-hand experience following the protocol in our laboratory, as well as teaching hands-on tutorials for the past year, we have identified two broad classes of users. On the one hand, there is the user that goes through the learning curve and experiences the benefits of following the protocol, both at the personal level, and when collaborating with others. In words of some of the attendees to the tutorials, “it quickly pays off” to go through the learning process. On the other hand, the fact that a Popper pipeline is implemented *a priori* (i.e. before an article has written), and that creating reusable pipelines implies the use of new tools (such as Docker or Spack), its adoption is seen by some potential new users as a big paradigm shift. The main criticism from this set of people is that “there is no time” for a researcher or student to do things in this “radically new way”.

Given the above, we see an opportunity to develop new OSS technology to close the gap for those that still resist to take the leap. The main objective is to create a platform where it is *ridiculously easy* to both, implement and re-execute experimentation pipelines. Our target quantifiable goals are “push-button” repeatability (re-execute an existing experiment) and “no more than 10 minutes” to assemble the skeletal components of a new scientific exploration pipeline (a few clicks on a web-based GUI should suffice).

[2]Based on our first-hand experience, we spend at most the same amount of time implementing a pipeline; what changes is the approach, i.e. we write scripts to automate a pipeline instead of manually executing it. This, in turn, means that the more we re-execute a pipeline, and the more we reuse stages across distinct explorations, the more it pays off.

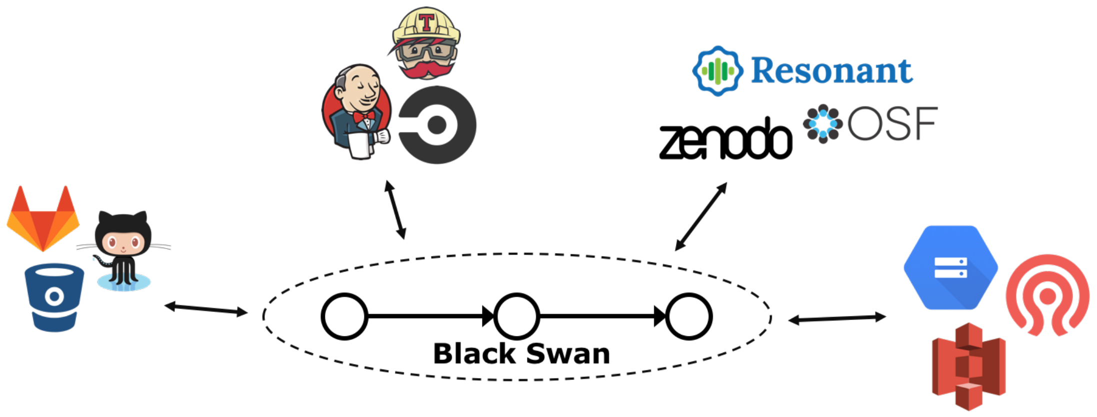


Figure 5.1: Black Swan will leverage existing services.

5.2 Black Swan

Once implemented, Black Swan will be a community-driven reproducibility platform that enables the agile delivery and validation of scientific insights. There are five main functional components of the platform.

External Service Integrations

One design principle is not to re-invent the wheel. That is, rather than re-implementing functionality found in other services or tools, Black Swan will have a pluggable mechanism so that it can integrate with these. A diagram of basic integrations is shown in Fig. 5.1. Version control services will store a pipeline's content; input and output will be version-controlled by connecting to dataset management services; CI services will continuously validate the integrity of a pipeline; and a database service will be used to store the history of executions for each pipeline, and as much environmental information as possible.

Automated Compliance Verification

Black Swan automates the process of verifying that a pipeline complies with the SEP criteria outlined previously, in a domain-agnostic way. Verification is done at runtime and is based on conventions of what the output of a pipeline looks like. For example, to verify that code and data repositories are being cloned, the output of a pipeline (what's get printed to `stdout`) is inspected to verify that repositories are being contacted (e.g. downloading from Github or Zenodo); the task of infrastructure allocation is expected to generate a `manifest.yml` (or `system.json`) file; explicit parametrization is verified by checking whether a `parameters.yml` file exists; and so on and so forth. By default, a pre-defined list of checks will be supported but the underlying mechanism will be extensible so more items are added in order to verify SEP compliance.

Pipeline Catalog and Pipeline Builder

In order to facilitate the adoption of the DevOps practice, the platform will incorporate a GUI component to allow users to visualize Popper pipelines and their stages. In particular, a *Pipeline Builder* will allow users to "mix and match" stages from an existing catalog of SEP compliant, community-maintained pipelines (Fig. **??**). The reusable catalog and the pipeline builder illustrate the importance that community will have in the success of Black Swan as an OSS project. Unless there is a community-wide effort in place, Black Swan will not succeed as a viable project; maintaining a pipeline by a single individual is too overwhelming.

Sketch of the pipeline builder GUI.

Differential Analysis

Whenever the validation stage of a SEP pipeline fails, we have found a black swan, i.e. the domain-specific expectations on results do not hold. Having access to the environmental information on which hardware and software was a pipeline re-executed on, Black Swan will provide facilities to automatically analyze and compare against previous successful executions in order to determine root causes of irreproducibility, or to aid researchers in finding them by incorporating GUI elements to support the visual inspection of differences.

Validation Dashboard

Borrowing ideas from [241], Black Swan will incorporate facilities to quickly visualize the status of a pipeline with respect to domain-specific validations. For failed validations, the dashboard will provide links to trigger differential analysis with respect to previous successful executions, as described above.

5.3 Use cases

We describe three use cases for which we envision Black Swan to be directly applicable.

Technology Transfer

Organizations with Research and Development (R&D) units such as tech companies and government-funded institutions (DOE labs, NASA, universities) spend significant amounts of resources transferring technology from R&D to production environments. Deploying an instance *Black Swan* internally (or using a public one) would allow organizations to streamline tech (and knowledge) transfers.

Research Curation

In the past decade, institutional libraries have invested a significant amount of resources in the creation of *Data Repositories*. These efforts are aimed at systematically managing the output of research. For example, EU's OpenAIRE[3] initiative is a catalog of software and data containing more than 80 million entries (each being a code repository or dataset). Similar efforts are underway in the US such as the Center for Open Science's Open Science Framework[4]. We see *Black Swan* complementing these efforts, since Popper enables the curation of research by enabling all these existing repositories to be easily executed and validated over time.

Self-validation of Academic Articles

A growing number of Computer Science conferences and journals incorporate an artifact evaluation process in which authors of an article submit an artifact description[5] (AD) that is tested by a committee, in order to verify that experiments presented in a paper can be re-executed by others. Black Swan can be leveraged to automatically test the reproducibility aspects of articles, assuming the pipeline(s) corresponding to an article are SEP compliant [25].

[3]http://openaire.eu/
[4]https://osf.io
[5]http://ctuning.org/ae/submission.html

5.4 Challenges

Managing changes to code using version-control systems; managing data with dataset management systems; continuously integrating (CI) and deploying (CD) software; all have become standard practices in OSS communities, not because of a fad but because of the quantifiable benefits that following best practices bring. To the contrary, in R&D settings, these practices are seen as a burden. The biggest challenge we face lies in changing the culture within organizations and teams; finding the right incentives so that these users can make the leap and adopt agile practices, so they can enjoy the benefits that come from embracing DevOps.

Bibliography

[1] T. Hey, S. Tansley, and K. Tolle, eds., *The Fourth Paradigm: Data-Intensive Scientific Discovery*, Microsoft Research, 2009.

[2] D.L. Donoho, A. Maleki, I.U. Rahman, M. Shahram, and V. Stodden, "Reproducible Research in Computational Harmonic Analysis," *Computing in Science & Engineering*, vol. 11, Jan. 2009, pp. 8–18. Available at: http://scitation.aip.org/content/aip/journal/cise/11/1/10.1109/MCSE.2009.15.

[3] T. Hoefler and R. Belli, "Scientific Benchmarking of Parallel Computing Systems: Twelve Ways to Tell the Masses when Reporting Performance Results," *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, New York, NY, USA: ACM, 2015, pp. 73:1–73:12. Available at: http://doi.acm.org/10.1145/2807591.2807644.

[4] P.J. Denning, "Is computer science science?" *Communications of the ACM*, vol. 48, 2005, pp. 27–31.

[5] G. Fursin, "Collective Mind: Cleaning up the research and experimentation mess in computer engineering using crowdsourcing, big data and machine learning," *arXiv:1308.2410 [cs, stat]*, Aug. 2013. Available at: http://arxiv.org/abs/1308.2410.

[6] C. Collberg and T.A. Proebsting, "Repeatability in Computer Systems Research," *Communications of the ACM*, vol. 59, Feb. 2016, pp. 62–69. Available at: http://doi.acm.org/10.1145/2812803.

[7] C. Larman and V.R. Basili, "Iterative and incremental developments. A brief history,"

Computer, vol. 36, Jun. 2003, pp. 47–56.

[8] S. McConnell, *Code Complete*, Microsoft Press, 1993.

[9] A. Hunt and D. Thomas, *The Pragmatic Programmer: From Journeyman to Master*, Addison-Wesley Professional, 1999.

[10] R.C. Martin, *Clean Code: A Handbook of Agile Software Craftsmanship*, Upper Saddle River, NJ: Prentice Hall, 2008.

[11] M. Hüttermann, *DevOps for Developers*, Apress, 2012.

[12] L. Bass, I. Weber, and L. Zhu, *DevOps: A Software Architect's Perspective*, Addison-Wesley Professional, 2015.

[13] G. Kim, J. Humble, P. Debois, and J. Willis, *The DevOps Handbook*, O'Reilly Media, 2016. Available at: http://shop.oreilly.com/product/9781942788003.do.

[14] J. Humble and D. Farley, *Continuous delivery: Reliable software releases through build, test, and deployment automation*, Pearson Education, 2010.

[15] M. Shahin, M.A. Babar, and L. Zhu, "Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices," *IEEE Access*, vol. 5, 2017, pp. 3909–3943.

[16] S. Davies, "Software development vs. Software delivery," *SAM'S SOFTWARE RAMBLINGS*, May. 2018. Available at: https://samueldavies.net/2018/05/22/software-development-vs-software-delivery/.

[17] L.E. Lwakatare, P. Kuvaja, and M. Oivo, "Dimensions of DevOps," *Agile Processes in Software Engineering and Extreme Programming*, C. Lassenius, T. Dingsøyr, and M. Paasivaara, eds., Springer International Publishing, 2015, pp. 212–217.

[18] M. Kersten, "Mining the Ground Truth of Enterprise Toolchains," *IEEE Software*, vol. 35, May. 2018, pp. 12–17.

[19] T. Laukkarinen, K. Kuusinen, and T. Mikkonen, "DevOps in Regulated Software Development: Case Medical Devices," *Proceedings of the 39th International Confer-*

ence on Software Engineering: New Ideas and Emerging Results Track, Piscataway, NJ, USA: IEEE Press, 2017, pp. 15–18. Available at: https://doi.org/10.1109/ICSE-NIER.2017.20.

[20] J.A. Morales, H. Yasar, and A. Volkman, "Implementing DevOps Practices in Highly Regulated Environments," *Proceedings of the International Workshop on Secure Software Engineering in DevOps and Agile Development 2018*, 2018.

[21] Grand View Research, "Development to Operations Market Size, Trends | DevOps Report, 2025," Mar. 2018. Available at: https://www.grandviewresearch.com/industry-analysis/development-to-operations-devops-market.

[22] B. Peterson, "Goldman Sachs is investing $20 million in $1.1 billion startup GitLab because the bank's engineers loved it so much: 'They were so happy as a customer'," *Business Insider*, Dec. 2018. Available at: https://www.businessinsider.com/goldman-sachs-gitlab-investment-2018-12.

[23] I. Jimenez, M. Sevilla, N. Watkins, C. Maltzahn, J. Lofstead, K. Mohror, A. Arpaci-Dusseau, and R. Arpaci-Dusseau, "Standing on the Shoulders of Giants by Managing Scientific Experiments Like Software," *USENIX; login*, vol. 41, Nov. 2016. Available at: https://www.usenix.org/publications/login/winter-2016-vol-41-no-4/jimenez.

[24] I. Jimenez, M. Sevilla, N. Watkins, C. Maltzahn, J. Lofstead, K. Mohror, A. Arpaci-Dusseau, and R. Arpaci-Dusseau, "The Popper Convention: Making Reproducible Systems Evaluation Practical," *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2017, pp. 1561–1570.

[25] I. Jimenez and C. Maltzahn, "Self-verifiable Experimentation Pipelines," Sep. 2018. Available at: https://zenodo.org/record/1414619.

[26] I. Jimenez and C. Maltzahn, *Spotting Black Swans With Ease: The Case for a Practical Reproducibility Platform*, Zenodo, 2018. Available at: https://zenodo.org/record/1488346.

[27] L. Dotson and S. Norris, "Developing a Campus-Wide Research Lifecycle: Perspectives from the University of Central Florida Libraries," *Faculty Scholarship and Creative*

Works, Jul. 2016. Available at: https://stars.library.ucf.edu/ucfscholar/34.

[28] The University of Winipeg Library, "Scholarly Communication and Publication Lifecycle," 2017. Available at: https://library.uwinnipeg.ca/scholarly-communication/index.html.

[29] A.M. Cox and W.W.T. Tam, "A critical analysis of lifecycle models of the research process and research data management," *Aslib Journal of Information Management*, vol. 70, 2018, pp. 142–157.

[30] A. Brett, M. Croucher, R. Haines, S. Hettrick, J. Hetherington, M. Stillwell, and C. Wyatt, *Research Software Engineers: State of the Nation Report 2017*, Zenodo, 2017. Available at: https://zenodo.org/record/495360.

[31] A. Trendowicz and J. Münch, "Factors Influencing Software Development ProductivityState-of-the-Art and Industrial Experiences," *Advances in computers*, vol. 77, 2009, pp. 185–241.

[32] A. Ahmed, S. Ahmad, N. Ehsan, E. Mirza, and S.Z. Sarwar, "Agile software development: Impact on productivity and quality," *Management of innovation and technology (ICMIT), 2010 IEEE international conference on*, IEEE, 2010, pp. 287–291.

[33] A. Bertolino, "Software Testing Research: Achievements, Challenges, Dreams," *2007 Future of Software Engineering*, Washington, DC, USA: IEEE Computer Society, 2007, pp. 85–103. Available at: http://dx.doi.org/10.1109/FOSE.2007.25.

[34] G.J. Myers, C. Sandler, and T. Badgett, *The Art of Software Testing*, John Wiley & Sons, 2011.

[35] N. Oreskes, K. Shrader-Frechette, and K. Belitz, "Verification, validation, and confirmation of numerical models in the earth sciences," *Science*, vol. 263, 1994, pp. 641–646.

[36] A. Saltelli, S. Tarantola, F. Campolongo, and M. Ratto, *Sensitivity analysis in practice: A guide to assessing scientific models*, John Wiley & Sons, 2004.

[37] W.L. Oberkampf and C.J. Roy, *Verification and validation in scientific computing*, Cambridge University Press, 2010.

[38] T. Mytkowicz, A. Diwan, M. Hauswirth, and P.F. Sweeney, “Producing Wrong Data Without Doing Anything Obviously Wrong!” *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*, New York, NY, USA: ACM, 2009, pp. 265–276. Available at: http://doi.acm.org/10.1145/1508244.1508275.

[39] T. Kalibera and R. Jones, “Rigorous benchmarking in reasonable time,” *ACM SIGPLAN Notices*, ACM, 2013, pp. 63–74.

[40] A. Goderis, U. Sattler, P. Lord, and C. Goble, “Seven bottlenecks to workflow reuse and repurposing,” *International Semantic Web Conference*, Springer, 2005, pp. 323–337.

[41] S.B. Davidson and J. Freire, “Provenance and Scientific Workflows: Challenges and Opportunities,” *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, New York, NY, USA: ACM, 2008, pp. 1345–1350. Available at: http://doi.acm.org/10.1145/1376616.1376772.

[42] M. Mattoso, J. Dias, K.A. Ocaña, E. Ogasawara, F. Costa, F. Horta, V. Silva, and D. de Oliveira, “Dynamic steering of HPC scientific workflows: A survey,” *Future Generation Computer Systems*, vol. 46, 2015, pp. 100–113.

[43] P. Ivie and D. Thain, “Reproducibility in Scientific Computing,” *ACM Comput. Surv.*, vol. 51, Jul. 2018, pp. 63:1–63:36. Available at: http://doi.acm.org/10.1145/3186266.

[44] K. Niemeyer, “Nature Editorial: If you want reproducible science, the software needs to be open source,” *Ars Technica*, vol. 26, 2012.

[45] R. Krebs, S. Spinner, N. Ahmed, and S. Kounev, “Resource usage control in multi-tenant applications,” *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*, IEEE, 2014, pp. 122–131.

[46] R.L. Sites, “Benchmarking "Hello, World!",” *Queue*, vol. 16, 2018, p. 10.

[47] J. Diaz, C. Munoz-Caro, and A. Nino, “A survey of parallel programming models and tools in the multi and many-core era,” *IEEE Transactions on parallel and distributed systems*, vol. 23, 2012, pp. 1369–1386.

[48] T. Hey and J. Hey, “E-Science and its implications for the library community,” *Library Hi Tech*, vol. 24, 2006, pp. 515–528.

[49] L. Yan and N. McKeown, “Learning Networking by Reproducing Research Results,” *SIGCOMM Comput. Commun. Rev.*, vol. 47, May. 2017, pp. 19–26. Available at: http://doi.acm.org/10.1145/3089262.3089266.

[50] M.D. Wilkinson, M. Dumontier, I.J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L.B. da Silva Santos, P.E. Bourne, J. Bouwman, A.J. Brookes, T. Clark, M. Crosas, I. Dillo, O. Dumon, S. Edmunds, C.T. Evelo, R. Finkers, A. Gonzalez-Beltran, A.J.G. Gray, P. Groth, C. Goble, J.S. Grethe, J. Heringa, P.A.C. ’t Hoen, R. Hooft, T. Kuhn, R. Kok, J. Kok, S.J. Lusher, M.E. Martone, A. Mons, A.L. Packer, B. Persson, P. Rocca-Serra, M. Roos, R. van Schaik, S.-A. Sansone, E. Schultes, T. Sengstag, T. Slater, G. Strawn, M.A. Swertz, M. Thompson, J. van der Lei, E. van Mulligen, J. Velterop, A. Waagmeester, P. Wittenburg, K. Wolstencroft, J. Zhao, and B. Mons, “The FAIR Guiding Principles for scientific data management and stewardship,” *Scientific Data*, vol. 3, Mar. 2016, p. 160018. Available at: https://www.nature.com/articles/sdata201618.

[51] J. Gray, “Tape is dead, disk is tape, flash is disk, RAM locality is king,” Jan. 2007.

[52] J. Gray and B. Fitzgerald, “Flash Disk Opportunity for Server Applications,” *Queue*, vol. 6, Jul. 2008, pp. 18–23. Available at: http://doi.acm.org/10.1145/1413254.1413261.

[53] A. Sandberg, L. Pareto, and T. Arts, “Agile collaborative research: Action principles for industry-academia collaboration,” *IEEE software*, vol. 28, 2011, pp. 74–83.

[54] S.R. Bradley, C.S. Hayter, and A.N. Link, “Models and Methods of University Technology Transfer,” *Foundations and Trends in Entrepreneurship*, vol. 9, May. 2013, pp. 571–650. Available at: https://nowpublishers.com/article/Details/ENT-048.

[55] T. Gorschek, P. Garre, S. Larsson, and C. Wohlin, “A model for technology transfer in practice,” *IEEE software*, vol. 23, 2006, pp. 88–95.

[56] G.K. Sandve, A. Nekrutenko, J. Taylor, and E. Hovig, “Ten Simple Rules for Reproducible Computational Research,” *PLoS Comput Biol*, vol. 9, Oct. 2013, p. e1003285. Available at: http://dx.doi.org/10.1371/journal.pcbi.1003285.

[57] G. Wilson, D.A. Aruliah, C.T. Brown, N.P.C. Hong, M. Davis, R.T. Guy, S.H. Haddock, K.D. Huff, I.M. Mitchell, and M.D. Plumbley, “Best practices for scientific computing,” *PLoS biology*, vol. 12, 2014, p. e1001745.

[58] J. Fehr, J. Heiland, C. Himpe, and J. Saak, “Best Practices for Replicability, Reproducibility and Reusability of Computer-Based Experiments Exemplified by Model Reduction Software,” *arXiv:1607.01191 [cs]*, Jul. 2016. Available at: http://arxiv.org/abs/1607.01191.

[59] J. Kitzes, D. Turek, and F. Deniz, *The practice of reproducible research: Case studies and lessons from the data-intensive sciences*, Univ of California Press, 2017.

[60] D. Nüst, C. Boettiger, and B. Marwick, “How to Read a Research Compendium,” *arXiv:1806.09525 [cs]*, Jun. 2018. Available at: http://arxiv.org/abs/1806.09525.

[61] J. Howison, “Retract bit-rotten publications: Aligning incentives for sustaining scientific software,” Jul. 2014. Available at: https://figshare.com/articles/Retract_bit_rotten_publications_Aligning_incentives_for_sustaining_scientific_software/1111632.

[62] M.-C. Gaudel, “Testing can be formal, too,” *Colloquium on Trees in Algebra and Programming*, Springer, 1995, pp. 82–96.

[63] P.C. Lane and F. Gobet, “Developing reproducible and comprehensible computational models,” *Artificial Intelligence*, vol. 144, 2003, pp. 251–263.

[64] D. Turi, P. Missier, C. Goble, D. De Roure, and T. Oinn, “Taverna workflows: Syntax and semantics,” *E-Science and Grid Computing, IEEE International Conference on*, IEEE, 2007, pp. 441–448.

[65] I. Jimenez, C. Maltzahn, J. Lofstead, A. Moody, K. Mohror, R. Arpaci-Dusseau, and A. Arpaci-Dusseau, “Characterizing and Reducing Cross-Platform Performance Variability Using OS-Level Virtualization,” *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, Chicago, USA: 2016, pp. 1077–1080.

[66] I. Jimenez, C. Maltzahn, J. Lofstead, A. Moody, K. Mohror, A. Arpaci-Dusseau, and R. Arpaci-Dusseau, “I Aver: Providing Declarative Experiment Specifications Facilitates the Evaluation of Computer Systems Research,” *TinyToCS*, vol. 4, 2016. Available at: http://tinytocs.ece.utexas.edu/papers/tinytocs4_paper_jimenez.pdf.

[67] I. Jimenez, “Clapp,” 2019.

[68] I. Jimenez, N. Watkins, M. Sevilla, J. Lofstead, and C. Maltzahn, “Quiho: Automated Performance Regression Testing Using Inferred Resource Utilization Profiles,” *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering*, New York, NY, USA: ACM, 2018, pp. 273–284. Available at: http://doi.acm.org/10.1145/3184407.3184422.

[69] A. Maricq, D. Duplyakin, I. Jimenez, C. Maltzahn, R. Stutsman, and R. Ricci, “Taming Performance Variability,” *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, Carlsbad, CA, USA: USENIX Association, 2018, pp. 409–425. Available at: http://dl.acm.org/citation.cfm?id=3291168.3291198.

[70] K.F. Fogel and M. Bar, *Open source development with CVS*, Coriolis Group Books, 2001.

[71] C.M. Pilato, B. Collins-Sussman, and B.W. Fitzpatrick, *Version Control with Subversion: Next Generation Open Source Version Control*, " O’Reilly Media, Inc.", 2008.

[72] L. Torvalds and J. Hamano, “Git: Fast version control system,” *URL http://git-scm.com*, 2010.

[73] G. King, “An Introduction to the Dataverse Network as an Infrastructure for Data Sharing,” *Sociological Methods & Research*, vol. 36, Jan. 2007, pp. 173–199. Available

at: http://smr.sagepub.com/content/36/2/173.

[74] H. Pampel, P. Vierkant, F. Scholze, R. Bertelmann, M. Kindling, J. Klump, H.-J. Goebelbecker, J. Gundlach, P. Schirmbacher, and U. Dierolf, "Making Research Data Repositories Visible: The re3data.org Registry," *PLoS ONE*, vol. 8, Nov. 2013, p. e78080. Available at: http://dx.doi.org/10.1371/journal.pone.0078080.

[75] A. Bhardwaj, S. Bhattacherjee, A. Chavan, A. Deshpande, A.J. Elmore, S. Madden, and A.G. Parameswaran, "DataHub: Collaborative Data Science & Dataset Version Management at Scale," *arXiv:1409.0798 [cs]*, Sep. 2014. Available at: http://arxiv.org/abs/1409.0798.

[76] L.H. Nielsen and T. Smith, "Zenodo Overview," Mar. 2014. Available at: https://zenodo.org/record/8428.

[77] D.D. Roure, C. Goble, and R. Stevens, "Designing the myExperiment Virtual Research Environment for the Social Sharing of Workflows," *IEEE International Conference on e-Science and Grid Computing*, 2007, pp. 603–610.

[78] V. Stodden, S. Miguez, and J. Seiler, "ResearchCompendia.Org: Cyberinfrastructure for Reproducibility and Collaboration in Computational Science," *Computing in Science & Engineering*, vol. 17, Jan. 2015, pp. 12–19. Available at: http://scitation.aip.org/content/aip/journal/cise/17/1/10.1109/MCSE.2015.18.

[79] E.D. Foster and A. Deardorff, "Open science framework (OSF)," *Journal of the Medical Library Association: JMLA*, vol. 105, 2017, p. 203.

[80] M. Rosenblum, "VMware's Virtual Platform," *Proceedings of hot chips*, 1999, pp. 185–196.

[81] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *ACM SIGOPS operating systems review*, ACM, 2003, pp. 164–177.

[82] F. Bellard, "QEMU, a fast and portable dynamic translator." *USENIX Annual Technical Conference, FREENIX Track*, 2005, p. 46.

[83] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, “Kvm: The Linux virtual machine monitor,” *Proceedings of the Linux symposium*, Dttawa, Dntorio, Canada, 2007, pp. 225–230.

[84] N. Huber, M. von Quast, M. Hauck, and S. Kounev, “Evaluating and Modeling Virtualization Performance Overhead for Cloud Environments.” *CLOSER*, 2011, pp. 563–573.

[85] B. Clark, T. Deshane, E. Dow, S. Evanchik, M. Finlayson, J. Herne, and J.N. Matthews, “Xen and the Art of Repeated Research,” *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, Berkeley, CA, USA: USENIX Association, 2004, pp. 47–47. Available at: http://dl.acm.org/citation.cfm?id=1247415.1247462.

[86] S. Soltesz, H. Pötzl, M.E. Fiuczynski, A. Bavier, and L. Peterson, “Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors,” *Proceedings of the 2Nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, New York, NY, USA: ACM, 2007, pp. 275–287. Available at: http://doi.acm.org/10.1145/1272996.1273025.

[87] C. Boettiger, “An introduction to Docker for reproducible research, with examples from the R environment,” *arXiv:1410.0846 [cs]*, Oct. 2014. Available at: http://arxiv.org/abs/1410.0846.

[88] P.J. Guo and M.I. Seltzer, “BURRITO: Wrapping Your Lab Notebook in Computational Infrastructure,” 2012. Available at: http://dash.harvard.edu/handle/1/9938866.

[89] C.C.R. Sanabria, O. Richard, and J. Emeras, “Reproducible Software Appliances for Experimentation,” *TRIDENTCOM - 9th International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities (2014)*, 2014. Available at: http://hal.inria.fr/hal-01064825.

[90] Q. Pham, S. Thaler, T. Malik, I. Foster, and B. Glavic, “Sharing and Reproducing Database Applications,” *Proc. VLDB Endow.*, vol. 8, Aug. 2015, pp. 1988–1991.

Available at: http://dx.doi.org/10.14778/2824032.2824118.

[91] A.P. Davison, M. Mattioni, D. Samarkanov, and B. Telenczuk, “Sumatra: A Toolkit for Reproducible Research,” *Implementing Reproducible Research*, 2014, p. 57.

[92] F. Chirigati, R. Rampin, D. Shasha, and J. Freire, “ReproZip: Computational Reproducibility with Ease,” *Proceedings of the 2016 ACM SIGMOD International Conference on Management of Data*, San Francisco, CA, USA: 2016.

[93] G. Fils, Z. Yuan, and T. Malik, “Sciunits: Reusable Research Objects,” *E-Science (e-Science), 2017 IEEE 13th International Conference on*, IEEE, 2017, pp. 374–383.

[94] F. Chirigati, D. Shasha, and J. Freire, “ReproZip: Using Provenance to Support Computational Reproducibility,” *Proceedings of the 5th USENIX Conference on Theory and Practice of Provenance*, Berkeley, CA, USA: USENIX Association, 2013, pp. 1–1. Available at: http://dl.acm.org/citation.cfm?id=2482613.2482614.

[95] G. Wilson, J. Bryan, K.A. Cranston, J. Kitzes, L. Nederbragt, and T.K. Teal, “Good enough practices in scientific computing,” *undefined*, 2017. Available at: /paper/Good-enough-practices-in-scientific-computing-Wilson-Bryan/2d878cb5a55c28384348d7efe23a0ae47d3a67a8.

[96] I.J. Taylor, E. Deelman, D.B. Gannon, and M. Shields, *Workflows for e-Science: Scientific workflows for grids*, Springer Publishing Company, Incorporated, 2014.

[97] A. Barker and J. van Hemert, “Scientific Workflow: A Survey and Research Directions,” *Parallel Processing and Applied Mathematics*, R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Wasniewski, eds., Springer Berlin Heidelberg, 2008, pp. 746–753. Available at: http://link.springer.com/chapter/10.1007/978-3-540-68111-3_78.

[98] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G.B. Berriman, J. Good, A. Laity, J.C. Jacob, and D.S. Katz, “Pegasus: A framework for mapping complex scientific workflows onto distributed systems,” *Sci. Program.*, vol. 13, Jul. 2005, pp. 219–237. Available at: http://dl.acm.org/citation.cfm?id=1239649.1239653.

[99] B.W. Boehm, *Software engineering economics*, Prentice-hall Englewood Cliffs (NJ),

1981.

[100] R.H. Thayer, S.C. Bailin, and M. Dorfman, *Software requirements engineerings*, IEEE Computer Society Press, 1997.

[101] R.S. Pressman, *Software engineering: A practitioner's approach*, Palgrave Macmillan, 2005.

[102] G. Elliott, *Global business information technology: An integrated systems approach*, Pearson Education, 2004.

[103] B. Henderson-Sellers, R. Due, I. Graham, and G. Collins, "Third generation OO processes: A critique of RUP and OPEN from a project management perspective," *Software Engineering Conference, 2000. APSEC 2000. Proceedings. Seventh Asia-Pacific*, IEEE, 2000, pp. 428–435.

[104] H.D. Benington, "Production of large computer programs," *Annals of the History of Computing*, vol. 5, 1983, pp. 350–361.

[105] B.W. Boehm, "A spiral model of software development and enhancement," *Computer*, vol. 21, 1988, pp. 61–72.

[106] P. Kruchten, *The rational unified process: An introduction*, Addison-Wesley Professional, 2004.

[107] P.A. Laplante and C.J. Neill, "The demise of the waterfall model is imminent," *Queue*, vol. 1, 2004, p. 10.

[108] K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, and R. Jeffries, "Manifesto for agile software development," 2001. Available at: http://agilemanifesto.org/.

[109] T. Dyb\a a and T. Dingsøyr, "Empirical studies of agile software development: A systematic review," *Information and software technology*, vol. 50, 2008, pp. 833–859.

[110] G. Lee and W. Xia, "Toward agile: An integrated analysis of quantitative and qualitative field data on software development agility," *Mis Quarterly*, vol. 34, 2010, pp. 87–114.

[111] T. Dingsøyr, S. Nerur, V. Balijepally, and N.B. Moe, *A decade of agile methodologies: Towards explaining agile software development*, Elsevier, 2012.

[112] K. Schwaber and M. Beedle, *Agile software development with Scrum*, Prentice Hall Upper Saddle River, 2002.

[113] K. Beck and E. Gamma, *Extreme programming explained: Embrace change*, addison-wesley professional, 2000.

[114] P. Simon, *The age of the platform: How Amazon, Apple, Facebook, and Google have redefined business*, BookBaby, 2011.

[115] U. Dolata, *Apple, Amazon, Google, Facebook, Microsoft: Market concentration - competition - innovation strategies*, University of Stuttgart, Institute for Social Sciences, Department of Organizational Sociology and Innovation Studies, 2017. Available at: https://econpapers.repec.org/paper/zbwstusoi/201701.htm.

[116] B. Beyer, C. Jones, J. Petoff, and N.R. Murphy, *Site Reliability Engineering: How Google Runs Production Systems*, " O'Reilly Media, Inc.", 2016.

[117] P. Ensor, "The functional silo syndrome," *AME Target*, vol. 16, 1988, p. 16.

[118] D.N. Blank-Edelman, *Seeking SRE: Conversations About Running Production Systems at Scale*, "O'Reilly Media, Inc.", 2018.

[119] M. Walls, *Building a DevOps culture*, " O'Reilly Media, Inc.", 2013.

[120] P. Debois, "DevOps Days Ghent," *DevopsDays. Retrieved*, vol. 31, 2009. Available at: http://www.devopsdays.org/events/2009-ghent.

[121] J. Robbins, "Operations is a competitive advantage," *O'Reilly Radar*, Oct. 2007. Available at: http://radar.oreilly.com/2007/10/operations-is-a-competitive-ad.html.

[122] J. Allspaw and P. Hammond, "10+ deploys per day: Dev and ops cooperation at Flickr," *Velocity: Web Performance and Operations Conference*, 2009.

[123] L.E. Lwakatare, P. Kuvaja, and M. Oivo, "Relationship of DevOps to Agile, Lean and Continuous Deployment," *International Conference on Product-Focused Software*

Process Improvement, Springer, 2016, pp. 399–415.

[124] D. Stahl, T. Martensson, and J. Bosch, “Continuous practices and devops: Beyond the buzz, what does it all mean?” *Software Engineering and Advanced Applications (SEAA), 2017 43rd Euromicro Conference on*, IEEE, 2017, pp. 440–448.

[125] W. De Kort, *DevOps on the Microsoft Stack*, Springer, 2016.

[126] Intellipaat, “What is DevOps? | DevOps Introduction | DevOps Tutorial For Beginners | Intellipaat,” Dec. 2013. Available at: https://www.youtube.com/watch?v=_I94-tJlovg.

[127] GOTO Conferences, “GOTO 2016 of High Performing Teams: Science Edition,” Apr. 2016. Available at: https://www.youtube.com/watch?v=EmDtmYwDs50.

[128] 3Pillar Global, “Death of the SysAdmin - Jonathan Rivers’ Talk at "Rise of the DevOps" in Cluj-Napoca,” Jan. 2017. Available at: https://www.youtube.com/watch?v=K7Lc7DMsj8w.

[129] AgileEE, “Talk "An agile introduction to DevOps" by Gil Zilberfeld for AgileEE 2017,” 2017. Available at: https://www.youtube.com/watch?v=totti8tWPkg.

[130] Coding Tech, “How Netflix Thinks of DevOps,” Mar. 2018. Available at: https://www.youtube.com/watch?v=UTKIT6STSVM.

[131] M. Hüttermann, “Infrastructure as code,” *DevOps for Developers*, Springer, 2012, pp. 135–156.

[132] K. Morris, *Infrastructure as code: Managing servers in the cloud*, " O’Reilly Media, Inc.", 2016.

[133] P. Baudiš, “Current Concepts in Version Control Systems,” *arXiv:1405.3496 [cs]*, May. 2014. Available at: http://arxiv.org/abs/1405.3496.

[134] A. Koc and A.U. Tansel, “A survey of version control systems,” *ICEME 2011*, 2011.

[135] A. Fiat and H. Kaplan, “Making Data Structures Confluently Persistent,” *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, Philadelphia,

PA, USA: Society for Industrial and Applied Mathematics, 2001, pp. 537–546. Available at: http://dl.acm.org/citation.cfm?id=365411.365528.

[136] E.D. Demaine, S. Langerman, and E. Price, “Confluently Persistent Tries for Efficient Version Control,” *Algorithm Theory 2008*, J. Gudmundsson, ed., Springer Berlin Heidelberg, 2008, pp. 160–172. Available at: http://link.springer.com/chapter/10.1007/978-3-540-69903-3_16.

[137] N. Tolia, M. Kozuch, M. Satyanarayanan, B. Karp, T.C. Bressoud, and A. Perrig, “Opportunistic Use of Content Addressable Storage for Distributed File Systems.” *USENIX Annual Technical Conference, General Track*, 2003, pp. 127–140.

[138] J.A. Smith, J.S. De Stefano Jr, J. Fetzko, C. Hollowell, H. Ito, M. Karasawa, J. Pryor, T. Rao, and W. Strecker-Kellogg, “Centralized fabric management using puppet, git, and GLPI,” *Journal of Physics: Conference Series*, IOP Publishing, 2012, p. 042056.

[139] T. De Nies, S. Magliacane, R. Verborgh, S. Coppens, P.T. Groth, E. Mannens, and R. Van de Walle, “Git2PROV: Exposing Version Control System Content as W3C PROV.” *International Semantic Web Conference (Posters & Demos)*, 2013, pp. 125–128.

[140] P. Nath, M.A. Kozuch, D.R. O’hallaron, J. Harkes, M. Satyanarayanan, N. Tolia, and M. Toups, “Design tradeoffs in applying content addressable storage to enterprise-scale systems based on virtual machines,” *management*, vol. 7, 2006, p. 20.

[141] A. Schaefer, M. Reichenbach, and D. Fey, “Continuous integration and automation for DevOps,” *IAENG Transactions on Engineering Technologies*, Springer, 2013, pp. 345–358.

[142] T.A. Limoncelli and D. Hughe, “LISA’11 ThemeDevOps: New Challenges, Proven Values,” *USENIX; login: Magazine*, vol. 36, 2011.

[143] R. Kazmi, D.N.A. Jawawi, R. Mohamad, and I. Ghani, “Effective Regression Test Case Selection: A Systematic Literature Review,” *ACM Comput. Surv.*, vol. 50, May. 2017, pp. 29:1–29:32. Available at: http://doi.acm.org/10.1145/3057269.

[144] D. Vohra, “Using Rolling Updates,” *Kubernetes Management Design Patterns*,

Springer, 2017, pp. 171–198.

[145] J.S. Ward and A. Barker, "Observing the clouds: A survey and taxonomy of cloud monitoring," *Journal of Cloud Computing*, vol. 3, 2014, p. 24.

[146] B.S. Farroha and D.L. Farroha, "A framework for managing mission needs, compliance, and trust in the DevOps environment," *Military Communications Conference (MILCOM), 2014 IEEE*, IEEE, 2014, pp. 288–293.

[147] M. Soni, "End to end automation on cloud with build pipeline: The case for DevOps in insurance industry, continuous integration, continuous testing, and continuous delivery," *Cloud Computing in Emerging Markets (CCEM), 2015 IEEE International Conference on*, IEEE, 2015, pp. 85–89.

[148] D. Olsen, *The lean product playbook: How to innovate with minimum viable products and rapid customer feedback*, John Wiley & Sons, 2015.

[149] S. Imran, M. Buchheit, B. Hollunder, and U. Schreier, "Tool Chains in Agile ALM Environments: A Short Introduction," *On the Move to Meaningful Internet Systems: OTM 2015 Workshops*, I. Ciuciu, H. Panetto, C. Debruyne, A. Aubry, P. Bollen, R. Valencia-García, A. Mishra, A. Fensel, and F. Ferri, eds., Springer International Publishing, 2015, pp. 371–380.

[150] R. Ricci and E. Eide, "Introducing CloudLab: Scientific Infrastructure for Advancing Cloud Architectures and Applications,"*;login:* vol. 39, pp. 36–38. Available at: http://www.usenix.org/publications/login/dec14/ricci.

[151] R. Bolze, F. Cappello, E. Caron, M. Daydé, F. Desprez, E. Jeannot, Y. Jégou, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, B. Quetier, O. Richard, E.-G. Talbi, and I. Touche, "Grid'5000: A Large Scale And Highly Reconfigurable Experimental Grid Testbed," *Int. J. High Perform. Comput. Appl.*, vol. 20, Nov. 2006, pp. 481–494. Available at: http://dx.doi.org/10.1177/1094342006070078.

[152] S. Webb, "Supporting more reliable results," *BioTechniques*, vol. 59, Aug. 2015, pp. 57–61.

[153] M.R. Munafò, B.A. Nosek, D.V. Bishop, K.S. Button, C.D. Chambers, N.P. Du

Sert, U. Simonsohn, E.-J. Wagenmakers, J.J. Ware, and J.P. Ioannidis, “A manifesto for reproducible science,” *Nature Human Behaviour*, vol. 1, 2017, p. 0021.

[154] J.B. Buckheit and D.L. Donoho, “WaveLab and Reproducible Research,” *Wavelets and Statistics*, A. Antoniadis and G. Oppenheim, eds., New York, NY: Springer New York, 1995, pp. 55–81. Available at: https://doi.org/10.1007/978-1-4612-2544-7_5.

[155] I. Jimenez, C. Maltzahn, J. Lofstead, M. Heroux, and K. Keahey, “Birds of a Feather session on Practical Reproducibility by Managing Experiments Like Software,” Nov. 2017. Available at: https://zenodo.org/record/1495244.

[156] D.G. Feitelson, “Experimental computer science: The need for a cultural change,” *Internet version: http://www. cs. huji. ac. il/∼ feit/papers/exp05. pdf*, 2006.

[157] P.J. Denning, “The science in computer science,” *Communications of the ACM*, vol. 56, 2013, pp. 35–38.

[158] J. Itkonen and M.V. Mäntylä, “Are test cases needed? Replicated comparison between exploratory and test-case-based software testing,” *Empirical Software Engineering*, vol. 19, 2014, pp. 303–342.

[159] M. Ceccato, A. Marchetto, L. Mariani, C.D. Nguyen, and P. Tonella, “Do automatically generated test cases make debugging easier? An experimental assessment of debugging effectiveness and efficiency,” *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 25, 2015, p. 5.

[160] V. Ivanov, A. Rogers, G. Succi, J. Yi, and V. Zorin, “What Do Software Engineers Care About? Gaps Between Research and Practice,” *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, New York, NY, USA: ACM, 2017, pp. 890–895. Available at: http://doi.acm.org/10.1145/3106237.3117778.

[161] S.M. Crook, A.P. Davison, and H.E. Plesser, “Learning from the past: Approaches for reproducibility in computational neuroscience,” *20 Years of Computational Neuroscience*, Springer, 2013, pp. 73–102.

[162] J. Mambretti, J. Chen, and F. Yeh, “Next Generation Clouds, the Chameleon Cloud Testbed, and Software Defined Networking (SDN),” *2015 International Conference*

on Cloud Computing Research and Innovation (ICCCRI), 2015, pp. 73–79.

[163] R. Cherrueau, D. Pertin, A. Simonet, A. Lebre, and M. Simonin, "Toward a Holistic Framework for Conducting Scientific Evaluations of OpenStack," *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CC-GRID)*, 2017, pp. 544–548.

[164] A.B. Yoo, M.A. Jette, and M. Grondona, "SLURM: Simple Linux Utility for Resource Management," *Job Scheduling Strategies for Parallel Processing*, D. Feitelson, L. Rudolph, and U. Schwiegelshohn, eds., Springer Berlin Heidelberg, 2003, pp. 44–60. Available at: http://link.springer.com/chapter/10.1007/10968987_3.

[165] W. McKinney, "Data structures for statistical computing in python," *Proceedings of the 9th Python in Science Conference*, Austin, TX, 2010, pp. 51–56.

[166] R. Core Team, "R: A language and environment for statistical computing," 2013.

[167] C.I. King, "Stress-ng," Oct. 2017. Available at: https://github.com/ColinIanKing/stress-ng.

[168] J. Treibig, G. Hager, and G. Wellein, "Likwid-bench: An extensible microbenchmarking platform for x86 multicore compute nodes," *Tools for High Performance Computing 2011*, Springer, 2012, pp. 27–36.

[169] J. Treibig, G. Hager, and G. Wellein, "Likwid: A lightweight performance-oriented tool suite for x86 multicore environments," *Parallel Processing Workshops (ICPPW), 2010 39th International Conference on*, IEEE, 2010, pp. 207–216.

[170] J.D. McCalpin, "Memory Bandwidth and Machine Balance in Current High Performance Computers," *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, Dec. 1995, pp. 19–25.

[171] I. Jimenez, C. Maltzahn, J. Lofstead, A. Moody, K. Mohror, R.H. Arpaci-Dusseau, and A. Arpaci-Dusseau, "The Role of Container Technology in Reproducible Computer Systems Research," *2015 IEEE International Conference on Cloud Engineering (IC2E)*, Tempe, AZ: 2015, pp. 379–385.

[172] S.A. Weil, S.A. Brandt, E.L. Miller, D.D.E. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," *Proceedings of the 7th symposium on Operating systems design and implementation*, Berkeley, CA, USA: USENIX Association, 2006, pp. 307–320. Available at: http://dl.acm.org/citation.cfm?id=1298455.1298485.

[173] Y. Sfakianakis, S. Mavridis, A. Papagiannis, S. Papageorgiou, M. Fountoulakis, M. Marazakis, and A. Bilas, "Vanguard: Increasing Server Efficiency via Workload Isolation in the Storage I/O Path," *Proceedings of the ACM Symposium on Cloud Computing*, New York, NY, USA: ACM, 2014, pp. 19:1–19:13. Available at: http://doi.acm.org.oca.ucsc.edu/10.1145/2670979.2670998.

[174] A.C. De Melo, "The new linux'perf'tools," *Slides from Linux Kongress*, 2010.

[175] N. Nethercote and J. Seward, "Valgrind: A framework for heavyweight dynamic binary instrumentation," *ACM Sigplan notices*, ACM, 2007, pp. 89–100.

[176] S. Williams, A. Waterman, and D. Patterson, "Roofline: An Insightful Visual Performance Model for Multicore Architectures," *Commun. ACM*, vol. 52, Apr. 2009, pp. 65–76. Available at: http://doi.acm.org/10.1145/1498765.1498785.

[177] Y.J. Lo, S. Williams, B. Van Straalen, T.J. Ligocki, M.J. Cordery, N.J. Wright, M.W. Hall, and L. Oliker, "Roofline model toolkit: A practical tool for architectural and program analysis," *International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*, Springer, 2014, pp. 129–148.

[178] A. Snavely, N. Wolter, and L. Carrington, "Modeling application performance by convolving machine signatures with application profiles," *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No.01EX538)*, 2001, pp. 149–156.

[179] D. Laws, "The Storage Engine: A Timeline of Milestones in Storage Technology," *Computer History Museum*, Nov. 2015. Available at: http://www.computerhistory.org/atchm/the-storage-engine-a-timeline-of-milestones-in-storage-technology/.

[180] E.F. Codd, "A relational model of data for large shared data banks," *Commun.*

ACM, vol. 13, 1970, pp. 377–387. Available at: http://portal.acm.org/citation.cfm?id=362685.

[181] A. Aji, F. Wang, H. Vo, R. Lee, Q. Liu, X. Zhang, and J. Saltz, “Hadoop gis: A high performance spatial data warehousing system over mapreduce,” *Proceedings of the VLDB Endowment*, vol. 6, 2013, pp. 1009–1020.

[182] J.E. Gonzalez, R.S. Xin, A. Dave, D. Crankshaw, M.J. Franklin, and I. Stoica, “GraphX: Graph Processing in a Distributed Dataflow Framework.” *OSDI*, 2014, pp. 599–613.

[183] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, “Hedera: Dynamic flow scheduling for data center networks.” *Nsdi*, 2010, pp. 89–92.

[184] D. Zhang, N. Jayasena, A. Lyashevsky, J.L. Greathouse, L. Xu, and M. Ignatowski, “TOP-PIM: Throughput-oriented programmable processing in memory,” *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing*, ACM, 2014, pp. 85–98.

[185] H. Kamp, “A theory of truth and semantic representation,” *Formal semantics-the essential readings*, 1981, pp. 189–222.

[186] H. Kamp and U. Reyle, *From discourse to logic: Introduction to modeltheoretic semantics of natural language, formal logic and discourse representation theory*, Springer Science & Business Media, 2013.

[187] I. Heim, “The semantics of definite and indefinite noun phrases,” 1982.

[188] R.A. Al-Zaidy, S.R. Choudhury, and C.L. Giles, “Automatic Summary Generation for Scientific Data Charts.” *AAAI Workshop: Scholarly Big Data*, 2016.

[189] R.A. Al-Zaidy and C.L. Giles, “A Machine Learning Approach for Semantic Structuring of Scientific Charts in Scholarly Documents.” *AAAI*, 2017, pp. 4644–4649.

[190] M. Cliche, D. Rosenberg, D. Madeka, and C. Yee, “Scatteract: Automated extraction of data from scatter plots,” *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, Springer, 2017, pp. 135–150.

[191] J. Poco and J. Heer, "Reverse-Engineering Visualizations: Recovering Visual Encodings from Chart Images," *Computer Graphics Forum*, Wiley Online Library, 2017, pp. 353–363.

[192] D. Jung, W. Kim, H. Song, J.-i. Hwang, B. Lee, B. Kim, and J. Seo, "Chart-sense: Interactive data extraction from chart images," *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, ACM, 2017, pp. 6706–6717.

[193] D. Dinakarpandian, Y. Lee, K. Vishwanath, and R. Lingambhotla, "MachineProse: An ontological framework for scientific assertions," *Journal of the American Medical Informatics Association*, vol. 13, 2006, pp. 220–232.

[194] H. Kilicoglu and S. Bergler, "Recognizing speculative language in biomedical research articles: A linguistically motivated perspective," *BMC bioinformatics*, vol. 9, 2008, p. S10.

[195] M. Light, X.Y. Qiu, and P. Srinivasan, "The language of bioscience: Facts, speculations, and statements in between," *HLT-NAACL 2004 Workshop: Linking Biological Literature, Ontologies and Databases*, 2004.

[196] M. Samwald and H. Stenzhorn, "Simple, ontology-based representation of biomedical statements through fine-granular entity tagging and new web standards," *The 12th Annual Bio-Ontologies Meeting*, 2009.

[197] A. Callahan and M. Dumontier, "Evaluating scientific hypotheses using the SPARQL inferencing notation," *Extended Semantic Web Conference*, Springer, 2012, pp. 647–658.

[198] J.H. Friedman, "Greedy Function Approximation: A Gradient Boosting Machine," *The Annals of Statistics*, vol. 29, 2001, pp. 1189–1232. Available at: http://www.jstor.org/stable/2699986.

[199] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, 2011, pp.

2825–2830. Available at: http://www.jmlr.org/papers/v12/pedregosa11a.html.

[200] T. Chen and C. Guestrin, “XGBoost: A Scalable Tree Boosting System,” *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, New York, NY, USA: ACM, 2016, pp. 785–794. Available at: http://doi.acm.org/10.1145/2939672.2939785.

[201] D.F. Specht, “A general regression neural network,” *IEEE transactions on neural networks*, vol. 2, 1991, pp. 568–576.

[202] F. Chollet, “Keras,” 2015. Available at: https://keras.io/.

[203] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, and M. Isard, “Tensorflow: A system for large-scale machine learning.” *OSDI*, 2016, pp. 265–283.

[204] J. Bergstra, D. Yamins, and D.D. Cox, “Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms,” *Proceedings of the 12th Python in Science Conference*, Citeseer, 2013, pp. 13–20.

[205] B. Komer, J. Bergstra, and C. Eliasmith, “Hyperopt-sklearn: Automatic hyper-parameter configuration for scikit-learn,” *ICML workshop on AutoML*, 2014, pp. 2825–2830.

[206] M.A. Heroux, D.W. Doerfler, P.S. Crozier, J.M. Willenbring, H.C. Edwards, A. Williams, M. Rajan, E.R. Keiter, H.K. Thornquist, and R.W. Numrich, “Improving performance via mini-applications,” *Sandia National Laboratories, Tech. Rep. SAND2009-5574*, vol. 3, 2009.

[207] M.A. Heroux, *Hpccg Solver Package*, Sandia National Laboratories, 2007. Available at: https://www.osti.gov/scitech/biblio/1230960.

[208] I. Karlin, J. Keasler, and J.R. Neely, *Lulesh 2.0 updates and changes*, Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2013.

[209] D.A. Bader and K. Madduri, “Design and Implementation of the HPCS Graph Analysis Benchmark on Symmetric Multiprocessors,” *High Performance Computing*

2005, Springer, Berlin, Heidelberg, 2005, pp. 465–476. Available at: https://link.springer.com/chapter/10.1007/11602569_48.

[210] K. Kira and L.A. Rendell, "A Practical Approach to Feature Selection," *Proceedings of the Ninth International Workshop on Machine Learning*, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1992, pp. 249–256. Available at: http://dl.acm.org/citation.cfm?id=645525.656966.

[211] D.A. Freedman, *Statistical Models: Theory and Practice*, Cambridge ; New York: Cambridge University Press, 2009.

[212] P. Prettenhofer and G. Louppe, "Gradient Boosted Regression Trees in Scikit-Learn," Feb. 2014. Available at: http://orbi.ulg.ac.be/handle/2268/163521.

[213] L. Breiman, J. Friedman, C.J. Stone, and R.A. Olshen, *Classification and Regression Trees*, Boca Raton: Chapman and Hall/CRC, 1984.

[214] K. Akbudak, E. Kayaaslan, and C. Aykanat, "Hypergraph Partitioning Based Models and Methods for Exploiting Cache Locality in Sparse Matrix-Vector Multiplication," *SIAM Journal on Scientific Computing*, vol. 35, Jan. 2013, pp. C237–C262. Available at: http://epubs.siam.org/doi/abs/10.1137/100813956.

[215] J. Zawodny, "Redis: Lightweight key/value store that goes the extra mile," *Linux Magazine*, vol. 79, 2009.

[216] S. van der Walt, S.C. Colbert, and G. Varoquaux, "The NumPy array: A structure for efficient numerical computation," *Computing in Science & Engineering*, vol. 13, 2011, pp. 22–30.

[217] S.E. Perl and W.E. Weihl, "Performance Assertion Checking," *Proceedings of the Fourteenth ACM Symposium on Operating Systems Principles*, New York, NY, USA: ACM, 1993, pp. 134–145. Available at: http://doi.acm.org/10.1145/168619.168630.

[218] M.D. Syer, Z.M. Jiang, M. Nagappan, A.E. Hassan, M. Nasser, and P. Flora, "Continuous Validation of Load Test Suites," *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering*, New York, NY, USA: ACM, 2014, pp. 259–270. Available at: http://doi.acm.org/10.1145/2568088.2568101.

[219] W. Shang, A.E. Hassan, M. Nasser, and P. Flora, "Automated Detection of Performance Regressions Using Regression Models on Clustered Performance Counters," *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, New York, NY, USA: ACM, 2015, pp. 15–26. Available at: http://doi.acm.org/10.1145/2668930.2688052.

[220] O. Ibidunmoye, F. Hernández-Rodriguez, and E. Elmroth, "Performance Anomaly Detection and Bottleneck Identification," *ACM Comput. Surv.*, vol. 48, Jul. 2015, pp. 4:1–4:35. Available at: http://doi.acm.org/10.1145/2791120.

[221] C. Heger, J. Happe, and R. Farahbod, "Automated Root Cause Isolation of Performance Regressions During Software Development," *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, New York, NY, USA: ACM, 2013, pp. 27–38. Available at: http://doi.acm.org/10.1145/2479871.2479879.

[222] S. Ghaith, M. Wang, P. Perry, and J. Murphy, "Profile-Based, Load-Independent Anomaly Detection and Analysis in Performance Regression Testing of Software Systems," *2013 17th European Conference on Software Maintenance and Reengineering (CSMR)*, 2013, pp. 379–383.

[223] D. Shen, Q. Luo, D. Poshyvanyk, and M. Grechanik, "Automating Performance Bottleneck Detection Using Search-based Application Profiling," *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, New York, NY, USA: ACM, 2015, pp. 270–281. Available at: http://doi.acm.org/10.1145/2771783.2771816.

[224] B. Gregg, "The Flame Graph," *Commun. ACM*, vol. 59, May. 2016, pp. 48–57. Available at: http://doi.acm.org/10.1145/2909476.

[225] C.P. Bezemer, J. Pouwelse, and B. Gregg, "Understanding software performance regressions using differential flame graphs," *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 2015, pp. 535–539.

[226] J. Chen and W. Shang, "An Exploratory Study of Performance Regression Introducing Code Changes," *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2017, pp. 341–352.

[227] G. Jung, G. Swint, J. Parekh, C. Pu, and A. Sahai, “Detecting Bottleneck in n-Tier IT Applications Through Analysis,” *Large Scale Management of Distributed Systems*, Springer, Berlin, Heidelberg, 2006, pp. 149–160. Available at: https://link.springer.com/chapter/10.1007/11907466_13.

[228] T.H. Nguyen, B. Adams, Z.M. Jiang, A.E. Hassan, M. Nasser, and P. Flora, “Automated Detection of Performance Regressions Using Statistical Process Control Techniques,” *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, New York, NY, USA: ACM, 2012, pp. 299–310. Available at: http://doi.acm.org/10.1145/2188286.2188344.

[229] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*, New York: Wiley, 1991.

[230] T. Kalibera, L. Bulej, and P. Tuma, “Benchmark precision and random initial state,” *Proceedings of the 2005 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2005)*, 2005, pp. 484–490.

[231] N.J. Wright, S. Smallen, C.M. Olschanowsky, J. Hayes, and A. Snavely, “Measuring and understanding variation in benchmark performance,” *DoD High Performance Computing Modernization Program Users Group Conference (HPCMP-UGC), 2009*, IEEE, 2009, pp. 438–443.

[232] A.B. De Oliveira, S. Fischmeister, A. Diwan, M. Hauswirth, and P.F. Sweeney, “Why you should care about quantile regression,” *ACM SIGPLAN Notices*, ACM, 2013, pp. 207–218.

[233] A. Maricq, D. Duplyakin, I. Jimenez, C. Maltzahn, R. Stutsman, and R. Ricci, “Data and analysis scripts for the OSDI 2018 paper "Taming Performance Variability",” Sep. 2018. Available at: https://zenodo.org/record/1435969.

[234] S. Kanev, J.P. Darago, K. Hazelwood, P. Ranganathan, T. Moseley, G.-Y. Wei, and D. Brooks, “Profiling a warehouse-scale computer,” *ACM SIGARCH Computer Architecture News*, ACM, 2015, pp. 158–169.

[235] C. Kozyrakis, A. Kansal, S. Sankar, and K. Vaid, “Server engineering insights for large-scale online services,” *IEEE micro*, 2010, pp. 8–19.

[236] M. Kambadur, T. Moseley, R. Hank, and M.A. Kim, “Measuring interference between live datacenter applications,” *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, IEEE Computer Society Press, 2012, p. 51.

[237] X. Zhang, E. Tune, R. Hagmann, R. Jnagal, V. Gokhale, and J. Wilkes, “CPI 2: CPU performance isolation for shared compute clusters,” *Proceedings of the 8th ACM European Conference on Computer Systems*, ACM, 2013, pp. 379–391.

[238] H.S. Gunawi, R.O. Suminto, R. Sears, C. Golliher, S. Sundararaman, X. Lin, T. Emami, W. Sheng, N. Bidokhti, and C. McCaffrey, “Fail-slow at scale: Evidence of hardware performance faults in large production systems,” *ACM Transactions on Storage (TOS)*, vol. 14, 2018, p. 23.

[239] L. Nussbaum, “Towards trustworthy testbeds thanks to throughout testing,” *Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2017 IEEE International*, IEEE, 2017, pp. 1571–1578.

[240] L. Nussbaum, “Testbeds support for reproducible research,” *Proceedings of the Reproducibility Workshop*, ACM, 2017, pp. 24–26.

[241] I. Jimenez, S. Hamedian, J. Lofstead, C. Maltzahn, K. Mohror, R. Arpaci-Dusseau, A. Arpaci-Dusseau, and R. Ricci, “PopperCI: Automated Reproducibility Validation,” *2017 IEEE INFOCOM International Workshop on Computer and Networking Experimental Research Using Testbeds (CNERT ’17)*, Atlanta, GA, USA: 2017.