PRINCIPLES AND TECHNIQUES FOR PERFORMANCE MANAGEMENT AND VALIDATION OF CLOUD HOSTED DISTRIBUTED APPLICATIONS

By

Yogesh Damodar Barve

Dissertation

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science

February 29, 2020

Nashville, Tennessee

Approved:

Aniruddha S. Gokhale, Ph.D.

Janos Sztipanovits, Ph.D.

Hongyang Sun, Ph.D.

Himanshu Neema, Ph.D.

Matthew Berger, Ph.D.

To my respected parents Damodar Barve and Rohini Barve,

my brother Mahesh Barve,

and

To my beloved wife Gauree Barve

ACKNOWLEDGMENTS

I would like to thank the following funding agencies for providing me with gracious finacial support which allowed me to work on my research ideas. The funding agencies include Air Force Research Lab (AFRL), National Science Foundation (NSF), Air Force Office of Scientific Research (AFOSR) and the National Institue of Standards and Technology (NIST).

I would like to thank my advisor Dr. Aniruddha Gokhale, for providing me with enormous opportunity to explore different frontiers of cloud computing research. My research and studies during this Ph.D. program would not have been possible without his constant support and guidance which I have received over the years. His willigness to give his valuable time for various research discussions, advice and research critiques is very much appreciated. He has been an enthusiastic supporter and champion of my research work.

I would like to express my deep gratitute to my co-advisor Dr. Janos Sztipanovits for his valuable and constructive suggestions and constant support of my research work. I am also grateful to Dr. Himanshu Neema, Dr. Hongyang Sun, and Dr. Matthew Berger for all the support, guidance and feedback which I received and also for serving on my dissertation committee. I am also thankful to Dr. Abhishek Dubey for insightful discussions, feedback and guidance during my research work.

I would like to thank my friends Anirban Bhattacharjee, Ajay Dev Chhokra, Robert Canady, Shweta Khare, Shunxing Bao, Travis Brummett, Zhuangwei Kang, and Ziran Min for their collaboration, feedback and encouragement. I would especially like to thank Dr. Faruk Caglar, Dr. Kyoungho An, Dr. Shashank Shekhar, Dr. Prithviraj Patil, and Dr. Subhav Pradhan for their mentorship during my doctoral studies.

Special thanks to my friends Bhim Kumar, Manisha Kumar, Swapnil Khaire and

Swapnaja Bhosle for all your thoughts, well wishes, and regular phone calls which always brought enormous joy and happiness.

Last but not the least, I would like to thank my family: my parents Damodar Barve and Rohini Barve, my brother Mahesh Barve and sister-in-law Ashwini Barve, and rest of my family for their unwavering support throughout this journey. Most importantly, I wish to thank my loving and supportive wife Gauree Barve for the relentless support and encouragement and my daughter Aishwarya who provide unending inspiration and excitement.

TABLE OF CONTENTS

Page

DEDICATION ii

ACKNOWLEDGMENTS iii

LIST OF TABLES ix

LIST OF FIGURES x

Chapter

I. Introduction 1

I.1. Emerging Trends 1
I.2. Research Challenges and Solution Requirements 2
I.2.1. Application-driven Challenges 2
I.2.2. Cloud-imposed Challenges 3
I.2.3. Accessibility Challenges 3
I.3. Research Scope and System Assumptions 5
I.4. Summary of Contributions 5
I.5. Dissertation Organization 8

II. Interference Aware Performance Modeling and Benchmarking 11

II.1. Introduction 11
II.2. Background and Literature Survey 14
II.2.1. Sources of Interference and Impact on Performance 14
II.2.2. Related Work 15
II.3. Solution Requirements and Proposed Approach 19
II.4. FECBench Methodology 20
II.4.1. FECBench Methodology and its Rationale 20
II.4.2. Benchmarking Isolated Characteristics of Applications 22
II.4.3. Application Clustering 23
II.4.4. Resource Utilization Profiling for Co-located Workloads 25
II.4.5. Building Resource Stressor Prediction Model 26
II.4.6. Design of Experiments (DoE) Specification 27
II.4.7. Creating the Stressor Knowledge Base 28
II.4.8. Building Performance Interference Prediction Model 28
II.5. System Architecture and Implementation 29
II.6. Experimental Validation 31
II.6.1. Experimental Setup 32

II.6.2. Validating the Resource Stressor Prediction Model . 32
II.6.3. Validating the Design Space Exploration Strategy . 33
II.6.4. Validating the Accuracy of the Performance Models 34
II.6.5. FECBench in Action: A Concrete Use Case 35
II.7. Conclusion 37

III. Addressing Accesibility Concerns in Performance Modeling 39

III.1. Introduction 39
III.2. Design and Implementation of UPSARA 41
III.2.1. Eliciting Challenges and Solution Needs 41
III.2.2. Architecture and Workflow 44
III.3. UPSARA's Domain-Specific Modeling Language (DSML) Design 46
III.3.1. Encoding UPSARA's Product-line Feature Model . 47
III.3.2. UPSARA Platform Meta-model 50
III.4. UPSARA's Generative Capabilities 54
III.4.1. Metric Monitoring Configuration Generation and Provisioning 56
III.4.2. Application Configuration Generation and Orchestration 58
III.5. Meeting The Requirements 58
III.6. Evaluating UPSARA via Use Cases 59
III.6.1. Case Study 1- Co-located Workload Performance Analysis 60
III.6.2. Case Study 2: Application Resource Utilization Modeling 61
III.6.3. Case Study 3: Studying the Impact of Application's Resource Configuration 62
III.7. Related Work 66
III.8. Conclusions 67

IV. Techniques for Generic Distributed Systems Validation Framework . 69

IV.1. Introduction 69
IV.2. Related Work 71
IV.3. Dimensions of Variability in the Development and Deployment of Distributed Systems 75
IV.3.1. Motivating Scenario 75
IV.3.2. Challenges and Requirements 77
IV.4. Design and Implementation of PADS 78
IV.4.1. Feature Model Representation 79
IV.4.2. Realizing The PADS Feature Model Using Model-driven Engineering 81
IV.4.3. Meeting the Requirements 86
IV.5. Framework Validation 89

IV.5.1. Extensibility of PADS 89
IV.5.2. Effectiveness of PADS 93
IV.5.3. Meeting the Requirements 94
IV.6. Concluding Remarks 95

V. Design Studio for Co-simulations 98

V.1. Introduction 98
V.2. Background 100
V.2.1. Co-Simulation Platform 101
V.2.2. Collaboration Platform 101
V.2.3. Collaborative Modeling Web-Bench 102
V.2.4. Cloud-Hosted Experimentation Platform 103
V.3. Design and Implementation of Design Studio 104
V.3.1. Design And Architecture 104
V.3.2. Modeling and Experimentation Workflow 110
V.4. Conclusions And Future Work 111

VI. Addressing Performance Issues in Distributed Co-simulations 112

VI.1. Introduction 112
VI.2. Motivation and Solution Requirements 114
VI.3. Overview of EXPPO 118
VI.4. Design Elements of EXPPO 119
VI.4.1. Performance Profiling of Federates 120
VI.4.2. Federation Resource Configuration Optimization . . 120
VI.4.3. Federation Machine Scheduling Heuristics 122
VI.5. Co-simulation as a Service Middleware Architecture 125
VI.6. Experimental Evaluation 126
VI.6.1. Experimental Setup 126
VI.6.2. Experimental Results 127
VI.7. Related Work 128
VI.8. Conclusion 129
VI.8.1. Lessons Learned & Future Work 130

VII. Research Outreach 132

VII.1. Introduction 132
VII.1.1. Complexities in Learning and Teaching Distributed Systems Algorithms 132
VII.1.2. Solution Approach and Organization of Chapter . . 133
VII.2. Related Work 135
VII.3. Design and Implementation of PADS 138
VII.3.1. Underlying Philosophy Behind PADS 139
VII.3.2. Feature Model Representation 141
VII.3.3. Realizing the PADS Feature Model using Model-driven Engineering 141

VII.3.4 Web Based Modeling Environment 142
VII.3.5 Roles and Responsibilities of PADS's Actors 143
VII.4 Runtime Architecture of PADS 145
VII.4.1 User Interaction Layer 145
VII.4.2 Backend Infrastructure Layer 148
VII.5 Framework Validation 151
VII.5.1 Preliminary User Study 151
VII.6 Concluding Remarks 162

VIII. Concluding Remarks 163

VIII.1 Summary of Contributions 163
VIII.2 List of Publications 165

REFERENCES 170

LIST OF TABLES

Table		Page
1.	Performance of Learned Models for Resource Stressors	33
2.	Hardware & Software Specification of Compute Server	60
3.	Increase in the number of generated code lines with increase in number of components	94

LIST OF FIGURES

Figure Page

1. CDF representation of prediction inference response times for the Inception RESNETv2 Keras model. We can see that the application's performance degrades when co-located with background applications as compared to its performance when running in isolation. 15

2. FECBench methodology. 21

3. Utilizations of different resources for each of the 106 applications from the benchmarking warehouse. Each application is represented by a unique number and is depicted along the horizontal axis. . . . 21

4. Radar charts illustrating the resource profiles for clusters 2, 4, 5, 7, 8 and 9. Resource pressure is higher when the vertices are closer to the edge of the radar chart on the resource axis. 24

5. Chart (a) shows the summation of the isolated L3 bandwidth resource pressures and the observed resource pressure on the system. Chart (b) shows the distribution of error from direct summation of isolated resource pressures of the applications. The mean absolute percentage error is 47%. 26

6. FECBench in action. The figure shows different components of the FECBench along with the manager host and the target physical hosts on which application profiling takes place. 30

7. Predicted resource utilizations across different resource types. Points concentrated along the diagonal indicate that the predicted values match the actual observed values. 33

8. Coverage of the predicted resource stressor design space exerted by the co-located application combinations available in the knowledge base. 34

9. Prediction accuracy in mean absolute percent error for `PMD`, `Canneal`, `InceptionResnetV2` applications when co-located with web search server from CloudSuite. 35

10. Cumulative distributions of prediction errors for the `PMD`, `Canneal`, `InceptionResnetV2` applications. 36

11. Use of FECBench in publish-process-subscribe showing: (a) impact of co-location on a topic's latency; (b) interference-aware placement of topics. 37

12. CDF of Response Time for Inception-ResNet v2 model using Keras with 4 Cores 39

13. High Level Overview of UPSARA 45

14. UPSARA Main Meta-model 48

15. Snippet of UPSARA Micro-metric Meta-model 51

16. Snippet of UPSARA Runtime Platform Meta-model 52

17. Snippet of UPSARA Scenario Meta-model 53

18. UPSARA Specialized Framework Meta-model 54

19. UPSARA Generative Process 55

20. Performance analysis of image processing service. a) Shows the latency distribution of the completion times of the image processing. b) Shows the dominant measurement metrics that are correlated with the increase in the latency of image processing 63

21. Normalized amount of context switches imposed by applications from the PARSEC, DaCaPo and Splash-2 benchmarks. 64

22. Normalized memory bandwidth utilization of applications from the PARSEC, DaCaPo and Splash-2 benchmarks. 64

23. Impact of different resource configurations on the applications' execution time. Variability in the execution time is due to scaling up in the configuration of the resources assigned to the application. . . . 65

24. Commonality and Variability in Algorithm Design and Deployment Workflow 76

25. Feature Model Diagram of Playground of Algorithms for Distributed Systems (PADS) Framework 79

26. Model-based Process for Distributed Algorithm Demonstration and Deployment 82

27. Meta-Model of Playground of Algorithms for Distributed Systems (PADS) Framework 83

28. Meta-Model of Network Actors 84

29. Meta-Model of DataRateChannel representing the communication characteristics 85

30. Model example 87

31. Model of target algorithm selection 88

32. Meta-Model of BitTorrent Algorithm 90

33. Metamodel of BitTorrent Network Connection 91

34. Model of BitTorrent Algorithm 92

35. Screenshot of the section of code generated by GME interpreter . . 93

36. Stakeholders in a Mixed Energy Smart Grid System 99

37. Overview of the Cloud based Modeling and Co-simulation of Mixed Electrical Energy Systems 105

38. Sequence diagram showcasing modeling and experimentation activity in design studio 109

39. Performance visualization of an example federation with three federates. 115

40. (a) Performance of the federate running the PARSEC Freqmine application using 8 threads for different resource configuration selections. The performance improves when assigned more cores. (b) Performance of five PARSEC benchmark applications for different resource configuration selections. The performance improves when assigned more cores. 116

41. (a) Workflow of EXPPO illustrating the connections between different components of the system. (b) Profiling code snippet in Java language generated leveraging the MDE techniques. 118

42. Co-simulation-as-a-Service 125

43. (a) Execution time for the EXPPO resource configuration compared to other strategies. (b) Cost for the EXPPO resource configuration compared to other strategies. 128

44. Feature Model Diagram of Playground of Algorithms for Distributed Systems (PADS) Framework 138

45. Model-based Process for Distributed Algorithm Demonstration and Deployment 144

46. Cloud based architecture of PADS framework 146

47. Runtime Experiment Webviewer illustrating Omnet++ simulator accessed through noVNC client 148

48. Network Topology One for user studies 153

49. Network Topology Two for user studies 154

50. Source code generated for user study network topology 158

51. Survey response to Question 1: Compare time to complete tasks for Topology 1 and Topology 2 by completing Manually and then using PADS Framework 159

52. Survey response to Question 2: Did PADS help you to avoid manual syntax errors compared to writing of the topology file manually? . 159

53. Survey response to Question 3: How easy was it to use PADS? . . . 160

54. Survey response to Question 4: How likely are you to use PADS in future assignments/experimentation? 160

55. Survey response to Question 5: Is PADS useful in learning distributed systems algorithms? 161

CHAPTER I

INTRODUCTION

I.1 Emerging Trends

Cyber Physical Systems (CPS) are found in many domains such as electric grids, autonomous vehicles, transportation networks and often manifest as a composition of multi-domain subsystems comprising electric, networking, thermodynamics, physical and control systems. These CPS involve different types of distributed applications that support building automation and control, smart power grid, health-care, and industrial processes and which demand stringent quality of service (QoS) requirements from their hosting platforms. Assuring that these QoS requirements are met and that the systems are safe and trustworthy is a complex problem and often involves large-scale simulations that integrate tools from multiple domains in different configurations. Thus, co-simulation is an attractive option for interlinking such multiple simulators to simulate higher-level, complex system behaviors.

Cloud computing offers an attractive platform for running such QoS-sensitive applications as is evident from the continual migration of applications, such as map-reduce, distributed co-simulations, and distributed machine learning to the cloud. Despite this promise, many challenges involving operationalization issues, such as performance variability, system monitoring and resource orchestration and deployment continue to persist for running distributed applications in cloud computing environments. Furthermore, as the variety and number of such applications increases, developers are faced with a lack of accessibility, management and validation solutions that ease application development and provisioning, and ensure the correctness of the deployed applications. Finally, the cost budget for executing these applications in the cloud dictates the broader usage of these systems. This doctoral dissertation is

concerned with addressing these challenges. In the following, we outline the key challenges associated with validating and running distributed applications in the cloud, and meeting their QoS needs.

I.2 Research Challenges and Solution Requirements

We now describe the different dimensions of challenges we address in this doctoral research and solution requirements.

I.2.1 Application-driven Challenges

- *Straggler Mitigation*: Distributed applications that are made up of independent communicating tasks and are deployed on distributed resources can often face the straggler problem. For example, co-simulations are a type of distributed applications comprising a group of simulators which coordinate and perform computations for a larger overall simulation activity. In time-stepped distributed co-simulations, all the participating entities wait on each other before progressing to the next computation step. This is characterized as the bulk-synchronous-parallel (BSP) execution paradigm [148]. In the BSP computation model, if some of the participants have a slower execution speed compared to the rest of the participants due to more utilized resources on which the tasks are deployed or more complex computations or both, the progress of the simulation as a whole is dictated by the slowest progressing participant. In these scenarios, we need to capture this execution behavior either apriori or during runtime so as to mitigate the slow down caused due to such straggler participants. However, doing so is difficult as the participating entities are not necessarily executing on a single host server, but rather are distributed across a set of machines. Secondly, there is a need for monitoring and capturing the performance traces for such large-scale distributed applications, which is a non-trivial task.

I.2.2 Cloud-imposed Challenges

- *Multi-tenant Cloud Environment*: In a cloud computing environment, resources are shared among various applications hosted by different providers. Thus, issues such as application interference [57] can affect the application performance. Host system overbooking and overloads can also affect the execution makespan of the applications thereby degrading the overall application performance. This unpredictability and variability in performance of distributed applications can result in violation of the application's QoS requirements.

- *Deployment Concerns*: Resource assignment plays an important role in the performance of the application. For example, in BSP style applications, ad hoc resource assignments can have an adverse effect on the wait-time between the fastest and slowest performing tasks in a given computation step of the co-simulation. This can potentially lead to longer completion times for the tasks and inadvertently impact the performance of the application. However, choosing the right set of resources for distributed applications is a non-trivial activity. Moreover, the user's cost budget also needs to be taken into account when scheduling resources for these applications.

I.2.3 Accessibility Challenges

- *Benchmarking and performance modeling*: Understanding the performance of applications deployed in the cloud environments requires an understanding of how applications perform when subjected to different system stress conditions. Monitoring the application performance and creating performance profiles allows one to estimate the behavior of application when running on such runtime platforms. However, there is a general lack of tools and techniques to conduct a systematic performance interference study. Further, creating a resource stress is also challenging for end users, who may not be an domain expert in

developing stressors on the underlying platform, however, need such stressors for the performance studies. Finally, with increasing heterogeneity of hardware platforms, the information related to commonality of monitoring probes and variability in the available monitoring metrics for these platforms can become unwieldy thereby increasing the complexity in configuration and deployment of this monitoring infrastructure for the end user. This in turn can lead to end users not capturing all the required monitoring metrics which are needed to build performance models of the distributed applications. The constructed performance model may not give high confidence outputs due to these missing metric information.

- *Domain-specific orchestration*: With the growing heterogeneity at the hardware architecture layer and increasing number of deployment platforms, accessibility of tools becomes a major challenge. Accidental complexities can manifest during the configuration, deployment and monitoring of the distributed applications on such platforms. These challenges if not addressed can potentially lead to unintended performance bottlenecks and bugs which could lead to cascades of failures in the distributed applications.

- *Rapid validation capabilities*: With the growing complexity of the distributed systems and applications, there is a need for studying and understanding the principles and algorithms which are at play in such systems, and validate the different system properties. Implementing and running these distributed systems algorithms can help in this activity. However, existing tools have a very high entry to barrier. Thus, users may find it difficult to test and validate these algorithms rapidly.

I.3 Research Scope and System Assumptions

Scope: In this dissertation we focus on distributed applications architected as a set of computational units executing computations in parallel to solve a given problem. These applications can be located on a single machine or a set of distributed machines in a cloud or potentially fog/edge computing environments which also support multi-tenancy. To assist humans in conducting performance benchmarking and validations, we leverage machine-assisted technologies that alleviate pain points in the deployment and orchestration of software artifacts. Furthermore, to allow for resource selection that enables performance- and cost-aware deployment of distributed co-simulations, which is a specific class of distributed systems that we consider, we leverage recommender systems that help select appropriate resources for the co-simulations.

Assumptions: In our study we have assumed that the network conditions in which the distributed applications are executing are stable. The tasks in the co-simulation experiments are assumed to follow the bulk synchronous parallel computations model and have nearly fixed wall-clock execution time for every iteration of the task. We assume that the co-simulations are deployed in shared computing resource nodes. We assume that the QoS metrics can be measured for both the applications and the runtime system. We also assume that there exists tools which allows for creating domain-specific modeling languages and simulation runtimes on which the study can be performed.

I.4 Summary of Contributions

To resolve the list of challenges described in I.2, in this doctoral research we make following concrete contributions.

Contribution 1: Performance interference-aware application modeling and benchmarking: To address the challenges discussed in I.2.2, in this work we

propose FECBench (Fog/Edge/Cloud Benchmarking), which is an open source framework comprising a set of 106 applications covering a wide range of application classes that guides users in building performance interference prediction models for their applications without incurring undue costs and efforts via the following contributions. First, we define a technique to build resource stressors that can stress multiple system resources all at once in a controlled manner, which help to gain insights into the impact of interference on the applications performance. Second, to overcome the need for exhaustive application profiling, FECBench intelligently uses the design of experiments (DoE) approach to enable users to build surrogate performance models of their services. Third, FECBench maintains an extensible knowledge base of application combinations that create resource stress across the multi-dimensional resources design space. Empirical results using real-world scenarios to validate the efficacy of FECBench shows that the predicted application performance using the surrogate models incurs a median error of only 7.6 percent across all tests, with 5.4 percent in the best case and 13.5 percent in the worst case. We describe FECBench details in chapter II.

Contribution 2: A Model-driven Approach for Performance Analysis of Cloud-hosted Applications: Similarly, to address challenges in I.2.2 and I.2.3, in this work we propose UPSARA - **U**nderstanding **P**erformance of **S**oftware **A**pplications and **R**untime System **A**nalysis, which is a model-driven generative framework that provides an extensible, lightweight and scalable performance monitoring, analysis and testing framework for cloud-hosted applications. UPSARA helps alleviate the accidental complexities in configuring the right resource monitoring and performance testing strategies for the underlying instrumentation frameworks used. We evaluate the effectiveness of UPSARA in the context of representative use cases highlighting its features and benefits. Chapter III presents more details on this contribution.

Contribution 3: Rapid design and validation platform for testing distributed systems algorithms: To address challenges I.2.3, in this work we propose PADS - **P**layground of **A**lgorithms for **D**istributed **S**ystems. We use the principles of software product lines (SPLs) and model-driven engineering and adopt the cloud platform to design an environment for rapid validation and testing of simulations in the cloud computing platform. The research contributions in PADS include the underlying feature model, the design of a domain specific modeling language that supports the feature model, and the generative capabilities that maximally automate the synthesis of experiments on cloud platforms. A prototype implementation of PADS is described to showcase a distributed systems algorithm illustrating a peer to peer file transfer algorithm based on BitTorrent, which shows the benefits of rapid deployment of the distributed systems algorithm. PADS contribution description is covered in chapter IV.

Contribution 4: Research outreach and broader impact: In this work, rather than create a point solution to address our challenges discussed in I.2.3 for distributed applications use-case, we found that we could also leverage our ongoing work to address challenges in teaching distributed systems concepts. To highlight the research outreach of our work, we present a user study as to how the PADS framework can be successfully leveraged to address complexities in teaching computer science concepts related to distributed systems. We also present the effectiveness of the PADS technology and additional enhancements done to the PADS technology to support larger user community. More details of this contribution is presented in chapter VII

Contribution 5: Addressing performance issues in co-simulations: Co-simulations, which is type of distributed applications, also suffer from the challenges as described in I.2.1,I.2.2 and , I.2.3. To address these challenges we make following specific contribution listed below.

Contribution 5.a: Co-simulation design studio: To address the challenges described in I.2.3 for rapid modeling and simulation of cyber-physical systems, in this work we propose a runtime platform for design and deployment of distributed simulations in the cloud computing platforms. We present the underlying workflow of design and deployment of the scientific experiments in the cloud computing environment with a mixed electrical energy systems as a motivational target scientific domain. We also present the underlying runtime architecture and deployment of distributed simulations leveraging the Docker container technology. Chapter V describes this contribution in more details.

Contribution 5.b: Straggler mitigation in distributed co-simulations: To address the challenge described in I.2.1, we present EXPPO - **E**Xecution **P**erformance **P**rofiling and **O**ptimization for Co-simulation-as-a-Service (CaaS) Platform, which addresses these challenges for BSP based computation discussed in I.2.1, by using execution performance profiling at each simulation execution step and for every simulator in a simulation. EXPPO uses the learned performance models from profiling in its simulation resource recommendation tool which solves an optimization problem to improve the execution performance of the co-simulation and also minimize the cost. Using an experimental testbed, the efficacy of EXPPO is validated to show the benefits of performance profiling and resource assignment in improving the execution runtimes of co-simulations and also minimizing the execution cost. Details of EXPPO are presented in chapter VI.

I.5 Dissertation Organization

The rest of the dissertation is organized as follows:

- Chapter II describes the performance modeling and benchmarking contributions supported by FECBench. This work appeared in *"Barve, Yogesh, Shashank Shekhar,Ajay Chhokra, Shweta Khare, Anirban Bhattacharjee, Zhuangwei Kang,*

Hongyang Sun and Aniruddha Gokhale. "FECBench: A Holistic Interference-aware Approach for Application Performance Modeling." In 2019 IEEE 11th International Conference on on Cloud Engineering (IC2E),IEEE, 2019."

- Chapter III describes the accessibility contributions made by the UPSARA framework. This work appeared in *"Barve, Yogesh, Shashank Shekhar, Shweta Khare, Anirban Bhattacharjee, and Aniruddha Gokhale. "UPSARA: A Model-Driven Approach for Performance Analysis of Cloud-Hosted Applications." In 2018 IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC), pp. 1-10. IEEE, 2018."*

- Chapter IV describes the PADS contributions that enable the rapid validation of cloud-based distributed applications. This work appeared in *"Barve, Yogesh D., Prithviraj Patil, and Aniruddha Gokhale. "A cloud-based immersive learning environment for distributed systems algorithms." In Computer Software and Applications Conference (COMPSAC), 2016 IEEE 40th Annual, vol. 1, pp. 754-763. IEEE, 2016."*

- Chapter V describes the infrastructure support used to host cloud-based distributed simulations and address their QoS needs. This work appeared in *"Barve, Yogesh, Himanshu Neema, Stephen Rees, and Janos Sztipanovits. "Towards a Design Studio for Collaborative Modeling and Co-Simulations of Mixed Electrical Energy Systems." In 2018 IEEE International Science of Smart City Operations and Platforms Engineering in Partnership with Global City Teams Challenge (SCOPE-GCTC), pp. 24-29. IEEE, 2018."*

- Chapter VII highlights the broader impact of the research contributions. This work appeared at *"Barve, Yogesh D., Prithviraj Patil, Anirban Bhattacharjee, and Aniruddha Gokhale. "Pads: Design and implementation of a cloud-based,*

immersive learning environment for distributed systems algorithms." IEEE Transactions on Emerging Topics in Computing 6, no. 1 (2018): 20-31."

- Chapter VI describes our approach for providing performance and cost aware deployment of co-simulations in cloud computing environment. This work is under-review for a conference submission.

- Chapter VIII summarizes the research contributions.

CHAPTER II

INTERFERENCE AWARE PERFORMANCE MODELING AND BENCHMARKING

II.1 Introduction

Context Multi-tenancy has become the hallmark of public cloud computing systems, where physical resources such as CPU, storage and networks are virtualized and shared among multiple different and co-located applications (i.e., tenants) to better utilize the physical resources. Although virtualization technologies such as virtual machines and containers allow cloud providers to increase the degree of multi-tenancy while still providing isolation of resources among the tenants, there exist non-partitionable physical resources such as the caches, TLBs, disk, and network I/O, which are susceptible to resource contention thereby causing adverse performance interference effects on the co-located tenants [90, 155]. Consequently, effective resource management solutions are required that can limit the impact of performance interference to acceptable levels such that the service level objectives (SLOs) of the applications can be maintained [38, 68, 69, 129].

Challenges Developing effective resource management solutions (e.g., schedulers) requires an accurate understanding of the target application's performance under different application co-location scenarios so that the impact of performance interference can be calibrated and accounted for in the solutions. Recent studies [119, 156] have built performance interference profiles for applications using a variety of resource utilization metrics. Since performance interference is caused due to the sharing of one or more non-partitionable resources, performance models for the application-under-study (i.e., the target application) that account for interference are developed

by co-locating them with a variety of *resource stressor applications* (i.e., those applications that put varying levels of pressure on the non-partionable resources) and recording the delivered performance to the target application.

Creating such performance models, however, requires the developer to expend significant efforts into application benchmarking and analyze application performance under varying levels of resource stress. Since the overall system utilization is a function of the stresses imposed on multiple types of resources in the system and the presence of multiple resources represents a multi-dimensional space, creating varying levels of resource stresses spanning this large design space is a difficult task. Effective resource management solutions, however, require application performance models that incorporate the impact of stresses on multiple resources all at once. Although existing resource stressors, such as *dummyload* or *stress-ng* [1], provide users with the control knobs to exert the desired level of stress on a resource, such as CPU or memory, these tools operate on only one resource at a time. Unfortunately, it is hard for users to define the right kinds of application workloads that will create the right levels of resource stress across the multi-dimensional space. All these problems are further exacerbated with the addition of fog and edge computing resources, which illustrate both increased heterogeneity and constraints on resources, and where the performance interference effects may be even more pronounced [50, 94].

Although, some frameworks/benchmarks exist that can assist in the building of the performance models, these tools remain mostly disparate and it takes a monumental effort on the part of the user to bring these disparate tools together into a single framework [89]. Even then, such a combined framework may not be easy to use. Moreover, a general lack of any systematic approach to conduct the performance modeling process will force the user to rely on *ad hoc* approaches, which hurts reproducibility and leads to reinvention of efforts [121], not to mention the possibility of the resulting models missing out on critical insights.

Beyond these challenges, one question still persists: *When is a performance model considered good enough such that it will enable effective resource management solutions?* In other words, how much application profiling is required to build these performance models? One strawman strategy to profile the application is to subject it to all possible resource stresses. However, such an approach will be time-consuming and even infeasible given the large number of combinations that can be executed on the different resource dimensions, the variety in the co-located application types, and their different possible workloads. Hence, there is a need for an intelligent application profiling strategy that minimizes the profiling effort and thereby the time and cost, while still providing sufficient coverage across all the resources that contribute to application performance interference. Unfortunately, there is a general lack of benchmarks and frameworks that can aid the user in developing these models.

Solution Approach To address these challenges, we present FECBench (Fog/Edge/-Cloud Benchmarking), an open source framework comprising a set of 106 applications that cover a wide range of application classes to guide providers in building performance models for their services without incurring undue costs and efforts. The framework can then be used to predict interference levels and make effective resource management decisions. Specifically, through the design of FECBench, we make the following contributions:

1. FECBench builds resource stressors that can stress multiple system resources all at once in a controlled manner. These resource stressors help in understanding the impact of interference effects on an application's performance.

2. To overcome the need for exhaustive application profiling, FECBench intelligently uses the design of experiments (DoE) approach to enable developers in building surrogate performance models of their services.

3. FECBench maintains an extensible knowledge base of application combinations that create resource stresses across the multi-dimensional resources design space.

Empirical results using real-world scenarios for validating the efficacy of FECBench show that the predicted application performance has a median error of only 7.6% across all test cases, with 5.4% in the best case and 13.5% in the worst case. A short poster version of this chapter describing initial work can be found in [30].

Chapter Organization The rest of the chapter is organized as follows: Section II.2 delves into the details of performance interference, and surveys the literature in this realm; Section II.3 elicits the key requirements for a solution such as FECBench; Section II.4 presents the design and implementation of FECBench, explaining how it meets the requirements outlined earlier; Section III.6 presents an extensive set of results validating the different features of FECBench; and finally Section II.7 presents concluding remarks discussing the implications of using FECBench and alluding to future work.

II.2 Background and Literature Survey

In this section we provide details on performance interference and its impact on application performance. We then present a survey of the literature in this area and the limitations of existing approaches, which motivate the key requirements of our FECBench solution.

II.2.1 Sources of Interference and Impact on Performance

Co-located applications on the same physical resources of a cloud platform will impose varying degrees of pressure (stress) on the underlying resources. When these resources are hard to partition or isolate, the contention for these resources will cause interference effects on the executing applications and degrade their performance. For compute-intensive applications, resources such as CPU core and Last Level Cache

(LLC) can cause interference. Similarly, for communication-intensive applications, resources such as memory bandwidth, disk I/O, and network can cause interference. For example, Figure 1 shows the performance degradation of an application that uses the Inception RESNETv2 deep learning model [13]. The figure illustrates a cumulative distribution function (CDF) for the 95th percentile response times of the application as its SLO with and without interference. Due to the significant difference in the observed response times, it is important for resource management solutions to incorporate the impact of interference to maintain application SLOs.

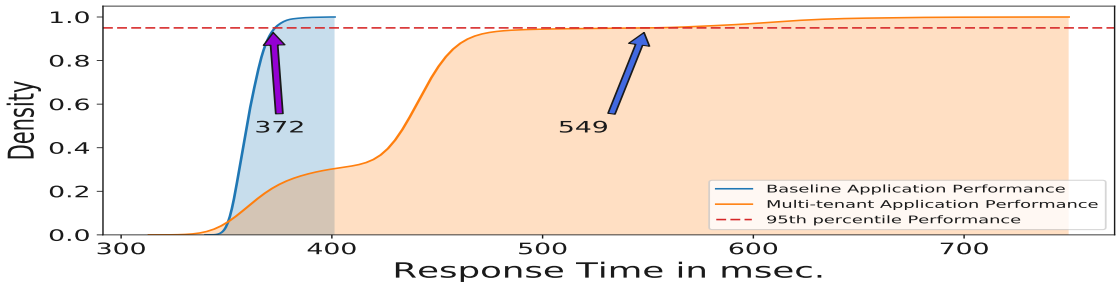

Figure 1: CDF representation of prediction inference response times for the Inception RESNETv2 Keras model. We can see that the application's performance degrades when co-located with background applications as compared to its performance when running in isolation.

II.2.2 Related Work

We now present prior efforts that focus on quantifying and modeling performance interference and classify these along three dimensions.

Interference Quantification Two fundamentally different approaches to quantifying performance interference have been reported in the literature. The *Bubble-Up* approach [107] measures *sensitivity* (i.e., impact of co-located applications on the target application) and *pressure* (i.e., impact of the target application on co-located applications) using a synthetic stressor application called *bubble*. The bubble generates a tunable amount of pressure on a given resource, such as memory or LLC. With

the pressure applied on a resource, the target application is executed simultaneously with co-located applications, and its performance metrics such as the completion time are measured. This experiment is repeated for different pressure levels in both the memory and the cache subsystems. Although this approach is effective, the bubble is limited to memory sub-system only and the approach is limited to two co-located applications. Yang et. al. [158] extended Bubble-Up to allow performance interference beyond two co-located applications and other shared resources such as network, I/O, cores, etc. The observed application performance degradation is used to construct sensitivity and pressure profiles which are used to determine if a given co-location will cause degradation in the performance of an application.

A different approach is presented in *DeepDive* [119], in which the performance interference is predicted based on the aggregate resource system utilization on the running system. Unlike Bubble-Up, where performance of an application is measured for stress levels independently on each resource, in DeepDive, application interference is measured by monitoring resource usage statistics of all co-located applications. In DeepDive, an application running inside a virtual machine is placed on an isolated physical machine and the resource utilization statistics are measured. The application is then migrated/placed on a physical machine by estimating the quality of interference level on the application and the co-located running applications. Inspired by the approach employed in DeepDive, FECBench leverages system resource metrics to build performance models for applications.

Interference-aware Predictive Modeling Building performance interference models and using them to predict the expected levels of interference for a given co-location configuration and workloads is important. Paragon [68] presents an interference-aware job scheduler in which an application's performance is predicted using collaborative filtering. The performance prediction model is built using performance data that is measured by subjecting the test application against individual resource stressors that

stress only one resource at a time. In comparison, FECBench takes into account the cumulative effect of all the resources to build an interference prediction model.

Zhao et. al. [162] studied the impact of co-located application performance for a single multi-core machine. They developed a piecewise regression model based on cache contention and bandwidth consumption of co-located applications. Similarly, their work captures the aggregate resource utilization of two subsystems, namely, cache contention and memory bandwidth, in determining the performance degradation. Our approach also considers disk and CPU resources in building prediction models and is not restricted to any hardware.

In DIAL [91], interference detection is accomplished using decision tree-based classifier to find the dominant source of resource contention. To quantify the resource interference impact on a webserver application's tail response, a queuing model is utilized to determine the application's response time under contention. To minimize the effects of interference, it proposed using a runtime controller responsible for dynamic load-balancing of queries from the webserver. Subramanian et. al. [143] presented an application slowdown model which estimates the application performance with high accuracy using cache access rate and memory bandwidth. However, the system was validated using a simulator and not on real hardware. In contrast, FECBench is geared towards real hardware.

The ESP project [112] uses a two-stage process for interference prediction. It first performs feature extraction, and then builds a regression model to predict performance interference. It creates separate models for each co-location groups. Also, its training data workload consists of all the possible applications that can run in the cluster. It then collects performance data for some combinations out of all the possible combinations to build the interference model. Similarly, Pythia [156] describes an approach for predicting resource contention given a set of co-located workloads. Both ESP and Pythia assume that they have *a priori* information of all possible running

workloads based on which an interference model is created for a new application. In comparison, FECBench relies on the performance metrics obtained when co-located with a fixed number of resource stressors and does not need to have prior information of all the running applications in the cluster.

Interference-related Synthetic Benchmarks One of the major roadblocks when investigating and building the performance interference modeling is the lack of representative benchmarking applications. Cuanta [81] built a synthetic cache loader that emulates pressure for varying tunable intensities on the LLC resource. It supports a Linux kernel module that invokes hypervisor system call to create the desired level of memory utilization. This kernel module resides inside a virtual machine. In contrast, our approach is non-intrusive and does not require any changes to the Linux kernel.

iBench [67] developed an extensive set of synthetic workloads that induce pressure on different resource subsystems. These workloads are built in a way to exert tunable utilization pressure on system resources such as L1, L2, iTLB, memory, LLC, disk, network, etc. in isolation. In [125], synthetic workloads were used to create pressure on network and CPU systems. Bubble-up [107], which was described earlier, is another effort in this category.

While these efforts made a step in the right direction, most existing approaches rely on manual tuning of the resource stressors to create the desired level of stress. As a result, prior approaches cannot adapt to changes in the underlying architecture. In contrast to earlier works, our approach finds application pairs and creates a resource stressor knowledge-base in an automated fashion. As a result, our approach can adapt to changes in the underlying architecture and can be reused. Moreover, with the help of design of experiments, FECBench reduces the profiling effort for the applications.

II.3 Solution Requirements and Proposed Approach

Based on the literature survey and unresolved challenges, we derive the following requirements for FECBench.

1. Benchmarking with Ease: Benchmarking and profiling applications can be a very tedious task because it involves configuration of probes on resources such as CPU, network or disk, and collection of many hardware- and application-specific performance metrics [89, 93, 139]. In this regard, tools such as `CollectD` [5] and `Systat` allow monitoring and collecting system metrics. Often, more than one tool may be required to collect the metrics of interest, which makes it hard for the user to integrate the tools. Moreover, dissemination of the monitored metrics in a timely manner to a centralized or distributed set of analysis engines must be supported to build performance interference models of the applications.

To address these challenges and to make the task of benchmarking easier and intuitive for the user, FECBench uses higher-level, intuitive abstractions in the form of domain-specific modeling [34, 41] and generative techniques to synthesize the generation of configurations, metrics collection and dissemination. Our recent work [32] describes these capabilities and hence it is not a focus of this chapter but we discuss this requirement for completeness sake.

2. Automated Construction of Resource Stressors: Tools like lookbusy and stress-ng can be utilized to create resource stress on CPU in a controlled tunable manner. Similarly, tools like iPerf can be utilized to create resource stress on the network resource. Despite this, there is a lack of open-source tools that can stress multiple resources simultaneously, also in a tunable manner. Moreover, some of the resource stressors are platform-specific, which hinders their applicability to heterogeneous platforms. Prior studies have presented design of stressors, which requires a deep understanding of the underlying hardware architecture and low-level resource

characteristics. Acquiring the skills to utilize these tools thus incurs a steep learning curve.

To address these concerns, Section II.4 A-G presents a process pipeline with offline and online stages that construct the multi-resource stressors in an automated fashion by leveraging machine learning techniques.

3. Minimizing the Prohibitive Profiling Cost: When building a performance interference model the user must profile the application's performance metrics against different configurations of resource utilization on the running system. However, since we have multiple resources, the resource utilization can be seen as a multi-dimensional design space. One approach to profiling is to exhaustively cover the entire design space and obtain the performance metrics for the application. However, the cost and time for executing these experiments will be very high. Thus, there is a need to significantly reduce the profiling effort while deriving good performance interference models.

To that end, we use the *design of experiments* (DoE) approach in Section II.4.6 to explore the multi-dimensional resource metrics using substantially lower experimental runs, for building the performance interference models.

II.4 FECBench Methodology

We now present FECBench and demonstrate its design methodology by building a performance interference model and explaining each step of its design.

II.4.1 FECBench Methodology and its Rationale

Figure 2 presents the FECBench process. The rationale for this process is described below and details of each step follow.

Recall that the goal of FECBench is to minimize the efforts for developers in building interference-aware performance models for their applications by providing

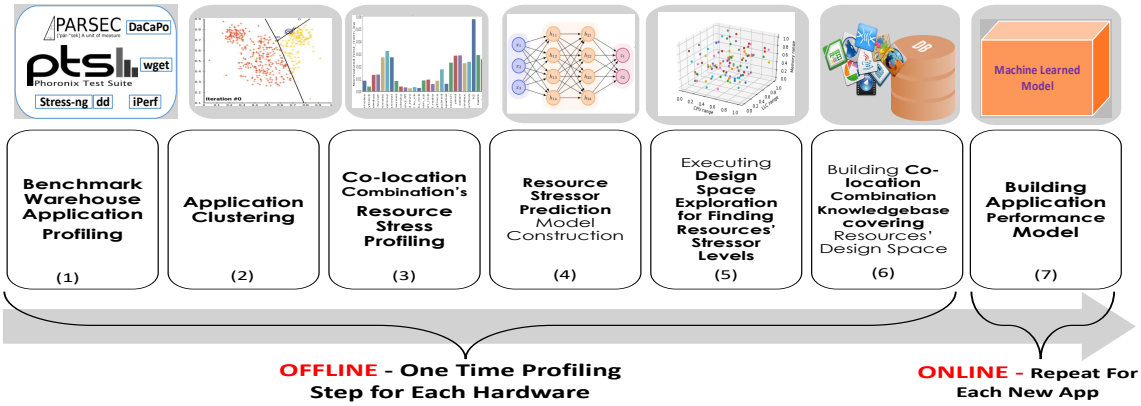

Figure 2: FECBench methodology.

them a reusable and extensible knowledge base. To that end, FECBench comprises an offline stage with a set of steps to create a knowledge base followed by an online stage. Developers can use the same offline stage process to further refine this knowledge base.

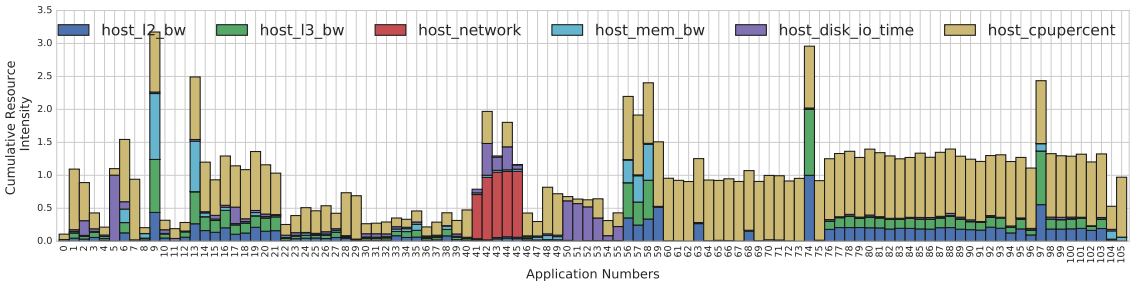

Figure 3: Utilizations of different resources for each of the 106 applications from the benchmarking warehouse. Each application is represented by a unique number and is depicted along the horizontal axis.

Accordingly, the first step (in §II.4.2) of the offline stage defines *Benchmark Warehouse* (BMW), which is a collection of resource utilization metrics obtained by executing a large number and variety of applications on a specific hardware and measuring the impact on each resource type independently. The second step (in §II.4.3) clusters these applications according to their similarity in how they stress individual resources. Clustering minimizes the unwieldiness stemming from the presence of a

large number of application types in the performance model building process. Next since we are interested in performance interference, the knowledge base must capture the stress on resources stemming from executing a combination of co-location patterns of applications belonging to the different clusters found in the earlier step (Step 3 in §II.4.4).

Using all this data, we define a resource stressor prediction model (Step 4 in §II.4.5), which can be used to predict the expected stress along the multi-dimensional resource space given a new co-location pattern. Since such a model building process itself may need exhaustive searching through every possible combination of resource stresses along the multi-dimensional resource space, we create surrogate models using design of experiments (DoE), specifically, the Latin Hypercube Sampling (LHS) approach (Step 5 in §II.4.6).

The reduced search strategies of Step 5 give rise to a knowledge base (Step 6 in §II.4.7), which is then used in the online stage (Step 7 in §II.4.8) that stresses a target application across different resource utilization regions from the design space to train a model for that target application and utilize it to predict its performance at runtime. The developer of a new application need only conduct Step 7 while leveraging all the previous steps. If a completely new hardware configuration is presented, the knowledge base must be updated by repeating all the steps of the offline stage. The steps are detailed next.

II.4.2 Benchmarking Isolated Characteristics of Applications

To build stressors that can stress different resources of a system, we first study the characteristics of different applications in isolation. We have profiled 106 applications from existing but disparate benchmarking suites, such as PARSEC [46],

DaCaPo [47], PHORONIX [17], STRESS-NG [1], as well as networked file-server applications, which collectively form our *benchmarking warehouse* (BMW). These applications represent a diverse range spanning from cloud computing to approximate computing workloads [159].

In this step, we profile the applications by running them in isolation on a system so as to document the resource utilization imposed by that application. Let A denote the set of all applications in BMW. For each application $a \in A$, we collect its runtime utilization metrics on the host system when run in isolation, including CPU, L2/L3 cache bandwidth, memory bandwidth, disk, network, etc. For a total number R of resources considered, the vector $U(a) = [u^{(1)}(a), u^{(2)}(a), \ldots, u^{(R)}(a)]$ is then logged in a database, where $u^{(r)}(a)$ denotes the utilization on a particular resource $r \in \{1, 2, \ldots, R\}$ when running the application. Figure 3 presents the resource utilization characteristics of 106 applications. The experiment host used is Intel(R) Xeon(R) CPU E5-2620 v4 machine with 16 physical cores. As we can see from the figure, the applications exhibit a high degree of coverage across the resource utilization spectrum for the different system resources.

II.4.3 Application Clustering

Given the large number of applications available in the BMW, it is likely that some of them exhibit similar characteristics with respect to the resource utilizations. For example, applications with numbers 80 and 81 in Figure 3 have similar utilizations with respect to CPU, L2 bandwidth and L3 bandwidth. This step performs clustering to identify those applications that share similar resource utilization characteristics. Moreover, application clustering allows us to select only a subset of applications from the BMW for the subsequent co-location resource utilization study. This helps to significantly reduce the number of application combinations that need to be profiled and tested.

Machine learning approaches, such as K-means clustering or Support Vector Method-based clustering, have been commonly used to find similarities in datasets [111]. In this study, we leverage the K-means algorithm to cluster all applications from the BWM in the R-dimensional space, where each dimension represents the utilization from a particular resource $r \in \{1, 2, \ldots, R\}$. Thus, each application $a \in A$ is represented by a point $U(a) = [u^{(1)}(a), u^{(2)}(a), \ldots, u^{(R)}(a)]$ in the R-dimensional space. We use the Silhouette algorithm [133] to determine the ideal number of clusters. For the considered 106 applications, running the algorithm leads to $K = 13$ clusters. Figure 4 shows the resource utilization characteristics for some of these clusters. As can be seen, Cluster 5 is L3, L2 and L3-system bandwidth-intensive. Similarly, Cluster 2 is memory bandwidth intensive. Cluster 9 shows high utilization pressures across network, disk and CPU.

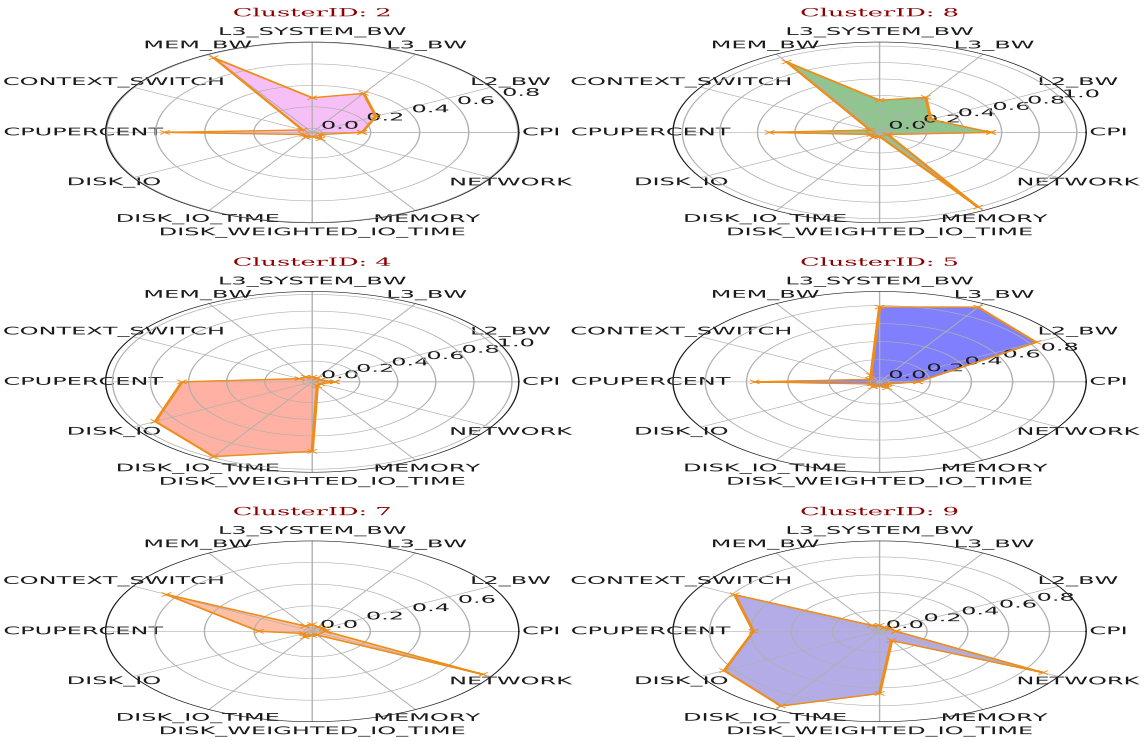

Figure 4: Radar charts illustrating the resource profiles for clusters 2, 4, 5, 7, 8 and 9. Resource pressure is higher when the vertices are closer to the edge of the radar chart on the resource axis.

II.4.4 Resource Utilization Profiling for Co-located Workloads

Since running a single application may not create the desired stress levels for the system resources, we are interested in finding those application mixes that together can create a more diverse set of resource stress levels. We first observe from our empirical experiments that the utilization of a resource on a system by running a set of co-located applications cannot be obtained by simply summing the resource utilizations of these applications when executed in isolation. This is validated by Figure 5, which shows that a direct summation of the isolated applications' L3 bandwidth utilizations incurs significant difference margins, with a mean absolute percent error of 47%. Similar behavior is also seen for other system resources.

This calls for profiling the resource utilization characteristics of different application co-location patterns. However, empirically running all application combinations is extremely time consuming. This step and the next together build a resource prediction model that determines the resource utilizations for any given application co-location pattern.

Before building a resource prediction model, we collect resource utilization data in this step by profiling a selection of application mixes, similar to the way we profiled a single application in Section II.4.2. Specifically, we pick an arbitrary application (e.g., the centroid) from each of K clusters and co-locate applications from different clusters to create different resource stressors. Let $d^{\max}$ denote the maximum number of co-located applications that are allowed in a run. This gives a total of $\sum_{d=1}^{d^{\max}} \binom{K}{d}$ application combinations.

Depending on the execution time needed to perform the profiling, one can choose a random subset of these combinations for building the prediction model. In our experiment, we have $K = 13$ and $d^{\max} = 8$ (since we are using a 16-core server and each application executes on 2 cores), which gives a total of 7,098 combinations. Among them, we profiled around 2,000 combinations for building the prediction model.

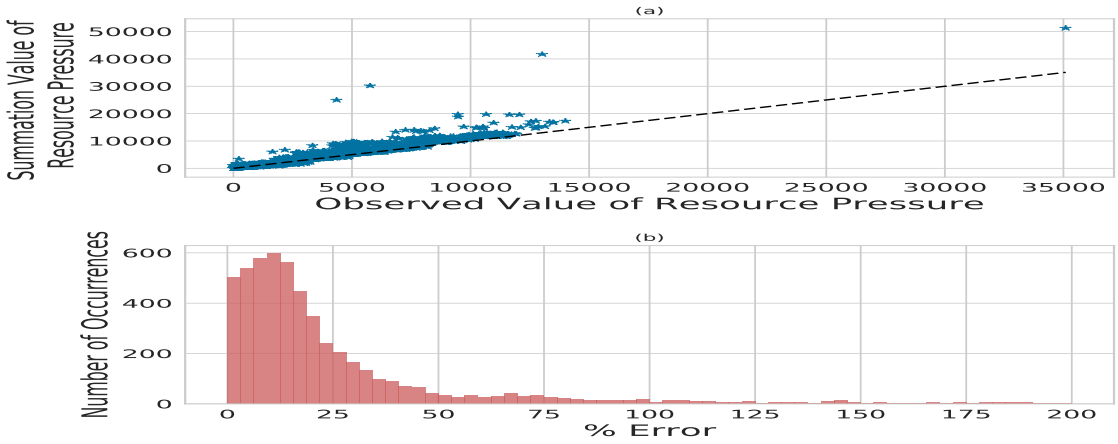

Figure 5: Chart (a) shows the summation of the isolated L3 bandwidth resource pressures and the observed resource pressure on the system. Chart (b) shows the distribution of error from direct summation of isolated resource pressures of the applications. The mean absolute percentage error is 47%.

II.4.5 Building Resource Stressor Prediction Model

Based on the resource utilization data collected from the last step, this step builds a resource stressor prediction model. We leverage random forest regression to determine the individual resource utilization given a set of co-located applications. Random forest regression is a widely used non-parametric regression technique for capturing non-linearity in the dataset. Unlike other methods, it performs well even with a large number of features and a relatively small training dataset, while providing an assessment of variable importance. Random forest is an ensemble-learning-based approach, where the results of multiple models, in this case, decision trees, are averaged to provide the final prediction. A decision tree recursively partitions a sample into increasingly more homogeneous groups up to a pre-defined depth. Terminal nodes in the tree then contain the final result. Random forest randomizes the creation of decision trees in two ways: 1) each decision tree is created from a random subset of input data; 2) each tree-partition is based on a random subset of input features.

For any application combination A, the following input features are used in the random forest prediction model:

- A K-dimensional vector $C = [c_1, c_2, \ldots, c_K]$, where each element c_i takes the value 1 if an application from the i-th cluster is selected in the combination A and 0 otherwise.

- A R-dimensional vector $U_+(A) = [u_+^{(1)}(A), u_+^{(2)}(A), \ldots, u_+^{(R)}(A)]$, where each element $u_+^{(r)}(A)$ represents the sum of individual utilizations for the r-th resource from all applications in A, i.e, $u_+^{(r)}(A) = \sum_{a \in A} u^{(r)}(a)$.

The output is another R-dimensional vector that predicts the utilization for all resources when executing the application combination A, i.e., $\tilde{U}(A) = [\tilde{u}^{(1)}(A), \tilde{u}^{(2)}(A), \ldots, \tilde{u}^{(R)}(A)]$. Thus, the resource stressor prediction model maps the input to the output via a prediction function f as follows:

$$\tilde{U}(A) \leftarrow f\big(C, U_+(A)\big)$$

II.4.6 Design of Experiments (DoE) Specification

To build a performance interference model for a target application when it is co-located with other applications, we need to measure its performance on a system that experiences different resource stress levels. Due to the large number of possible stress levels along multiple resource dimensions, it is not practical to test all of them. Therefore, we adopt the design of experiments (DoE) [58] approach by generating a small number of sample points in the multi-dimensional space that maximizes the coverage of the different resource utilizations.

To this end, we leverage the Latin Hypercube Sampling (LHS) method [58] that generates sampled regions across the R-dimensional resource utilization space. Specifically, LHS divides each resource dimension into M equally-spaced intervals, and then selects M sample intervals in the entire R-dimensional space that satisfies the Latin Hypercube property: each selected sample is the only one in each axis-aligned hyperplane that contains it. The LHS method has a clear advantage over random

sampling, which could potentially lead to selections of samples that are all clumped into a specific region. Moreover, the number M of samples in LHS does not grow with the number of dimensions. For a given choice of M, it generates a collection $H = \{h_1, h_2, \ldots, h_M\}$ of M hypercubes. Since each dimension represents the resource utilization of a corresponding resource in our case, its overall range is $[0, 1]$. Therefore, each generated hypercube $h_i \in H$ in a resource dimension r has the range $\left[\frac{x_i^{(r)}}{M}, \frac{x_i^{(r)}+1}{M}\right]$ for some $x_i^{(r)} \in \{0, 1, \ldots, M-1\}$. We refer interested readers to [104] for an in-depth explanation of the LHS method. In our experiment, we set $M = 300$.

II.4.7 Creating the Stressor Knowledge Base

We now create a knowledge-base of applications and their workload mixes that map to the different resource utilization levels as determined from the DoE exploration. Let $\mathcal{SA}$ denote the set of all application combinations generated in Section II.4.4. We consider every application combination $A \in \mathcal{SA}$ and use the resource stressor prediction model of Section II.4.5 to predict its utilization $\tilde{u}^{(r)}(A)$ for each individual resource $r \in \{1, 2, \ldots, R\}$. We then fill up each of the M hypercubes sampled in Section II.4.6 with the application combinations that belong to it. Specifically, for each application combination A, it is assigned to hypercube $h_i \in H$, if $\tilde{u}^{(r)}(A) \in \left[\frac{x_i^{(r)}-\delta}{M}, \frac{x_i^{(r)}+1+\delta}{M}\right]$ for all $1 \leq r \leq R$, where $\delta > 0$ is a tolerance parameter to extend the boundaries of the hypercubes to account for the inaccuracy of the stressor prediction model. In our experiment, we set $\delta = 0.1$.

II.4.8 Building Performance Interference Prediction Model

In the last step (which is an online step), a developer must construct a performance interference prediction model for a new target application b that is introduced for the first time onto the platform. The goal is to predict a specified QoS metric q for the target application when it is co-located with any set B of applications. To that end,

we leverage a regression-based Decision Tree model [126]. The input of the prediction model is a R-dimensional vector $U = [u^{(1)}, u^{(2)}, \ldots, u^{(R)}]$ showing the utilization of different system resources before the target application b is deployed. The output is the predicted QoS metric for the target application b, which we denote as $\tilde{q}(b, U)$, under the current system utilization U. Thus, the interference prediction model maps the input to the output via a prediction function g as follows:

$$\tilde{q}(b, U) \leftarrow g\big(U\big)$$

To build the regression model, the target application is executed under different resource stress levels identified by the design of experiments in Section II.4.6. For each of the M resource stress levels, the target application is executed along with a selected application combination from the knowledge base that corresponds to the desired resource stress level. To select the application combination, the closest application to the center of each hypercube is chosen. This selected application combination is first run on the platform. After a warm-up period, the target application is then deployed on the same platform, and its performance QoS metric is logged. This process is repeated for all the M resource stress levels. The results are used as training data to train the regression model above. In FECBench, we consider the response time, which is the computation time of the application, as the QoS metric used for latency-sensitive applications with soft real-time requirements.

II.5 System Architecture and Implementation

Figure 6 shows the different components of FECBench. There are two classes of nodes: *manager host* and *physical hosts*. Manager host is responsible for the management and orchestration of FECBench. We describe the numbered components in the figure below.

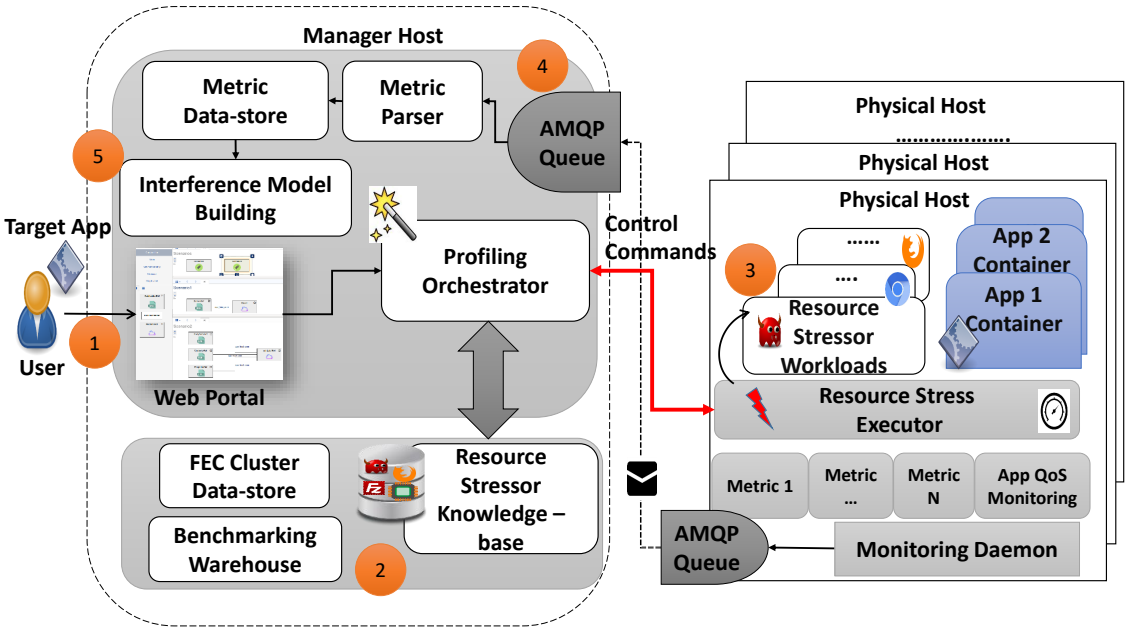

Figure 6: FECBench in action. The figure shows different components of the FECBench along with the manager host and the target physical hosts on which application profiling takes place.

In ①, the *Webportal* component at the manager host allows the user to interact with FECBench. The Webportal is built using a visual domain specific modeling language [35]. It allows the user to submit the target application whose performance interference model needs to be constructed. The Webportal initiates the profiling of the target application, and relays this information to the *Profiling orchestrator*, which then fetches the required information – resource stressors and system availability – from the FECBench system information block represented by ②. Components in ② comprise the FEC Cluster datastore, Resource Stressor Knowledgebase, and Benchmarking Warehouse. The FEC Cluster datastore consists of the current information about the cluster. Equipped with the required information, the Profiling orchestrator deploys the target application on the desired physical host, where the interference-aware profiling of the application takes place. In ③, the target application is subjected to various resource stressors as obtained from the stressor knowledge

base. The monitoring probes on the physical hosts monitor the system as well as the application QoS metrics and this information is relayed to the manager host. In ④, the desired metrics are parsed and data is stored in a time series datastore. In ⑤, FECBench constructs the interference-aware performance model of the target application.

The monitoring of the system performance metrics is done using the *CollectD* monitoring program. For the system metrics not supported by CollectD, custom Python based plugins are written that feed the metric data to the CollectD daemon. CollectD plugins for the Linux *perf* utility and *Likwid* monitoring tool [15] are written to monitor and log additional system metrics such as the cache-level and memory-level bandwidth information. Docker-based resource stressor containers have also been built. The monitored metrics are relayed in real time using AMQP message queues. Metric parsers for the gathered data are written using both Golang and Python languages. We use *Influxdb* to provide a time series database for storing the monitoring metrics [10]. For building performance models, we leverage the machine learning libraries provided by the *Scikit* library in Python [19].

II.6 Experimental Validation

This section validates the claims we made about FECBench. To that end, we demonstrate how FECBench enables savings in efforts in building the performance models of applications and their accuracy in making resource management decisions. We validate the individual steps of the FECBench process. We also present a concrete use case that leverages FECBench for interference-aware load balancing of topics on a publish-process-subscribe system.

II.6.1 Experimental Setup

We validated the FECBench claims for a specific hardware comprising an Intel(R) Xeon(R) CPU E5-2620 v4 compute node with 2.10 GHz CPU speed, 16 physical cores, and 32 GB memory. The software details are as follows: Ubuntu 16.04.3 64-bit, Collectd (v5.8.0.357.gd77088d), Linux Perf (v4.10.17) and Likwid Perf (v4.3.0). For the experiments, we configured the `scaling_governor` parameter of CPU frequency to `performance` mode to achieve the maximum performance.

II.6.2 Validating the Resource Stressor Prediction Model

To build and validate the resource stressor prediction model (Step 4 of FECBench), we first need to obtain a dataset that includes the resource utilizations under different application co-location scenarios (Step 3). In our experiment, we pin each application to 2 cores of the test node for a maximum of 8 co-located applications (since the node has 16 cores). Our offline profiling for Step 3 produced a dataset of about 2,000 data points, of which we used 80% for training, 10% for testing and the remaining 10% for validation. Table 1 illustrates the performance of the resource stressor prediction model. We see that the learned models have high accuracy for both the test and the trained dataset. We used the same accuracy measure, *coeffecient of determination (R^2)*, as in prior studies [112]. We observe an accuracy of 99.1% and 99.3% for the training and testing data, respectively, for memory bandwidth. The learned model also has low bias and variance since both testing and validation errors converge for most cases.

Figure 7 shows the accuracy of FECBench in predicting the actual resource utilizations for the co-located workloads. We see that the resource utilization predicted by FECBench are quite accurate as the bulk of the points fall close to the diagonal region of the chart with very few outliers.

Table 1: Performance of Learned Models for Resource Stressors

Feature	Test Accuracy	Train Accuracy	Validation Accuracy
MEM_BW	99.305	99.073	98.930
CPUPERCENT	89.896	88.776	88.889
MEMORY	98.310	98.346	86.143
L3_BW	99.085	98.839	98.037
NETWORK	99.663	99.665	99.673
L3_SYSTEM_BW	99.367	98.888	98.452
L2_BW	99.454	99.130	97.916
DISK_IO_TIME	88.002	88.464	88.519

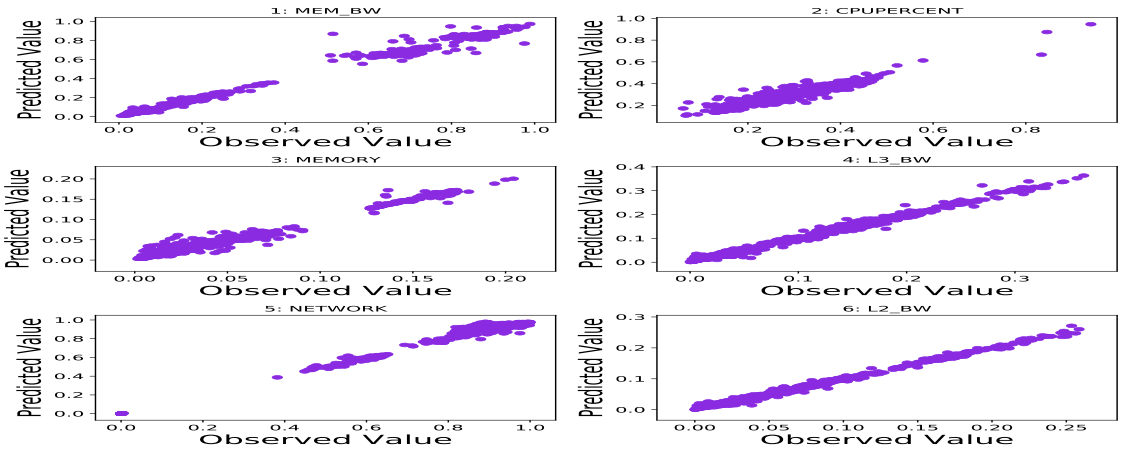

Figure 7: Predicted resource utilizations across different resource types. Points concentrated along the diagonal indicate that the predicted values match the actual observed values.

II.6.3 Validating the Design Space Exploration Strategy

For this experiment, the goal is to find the right application combinations that exert pressure in a tunable fashion along multiple resource dimensions (in our case, CPU, memory bandwidth and disk resources). We set the number of samples for the LHS strategy to 300 in the design of experiments, and got coverage for around 264 bins, i.e., 88%. To allow for easier visualization of the coverage, we project the three dimensional datapoints on a two dimensional scale as shown in Figure 8, demonstrating good coverage of the design space.

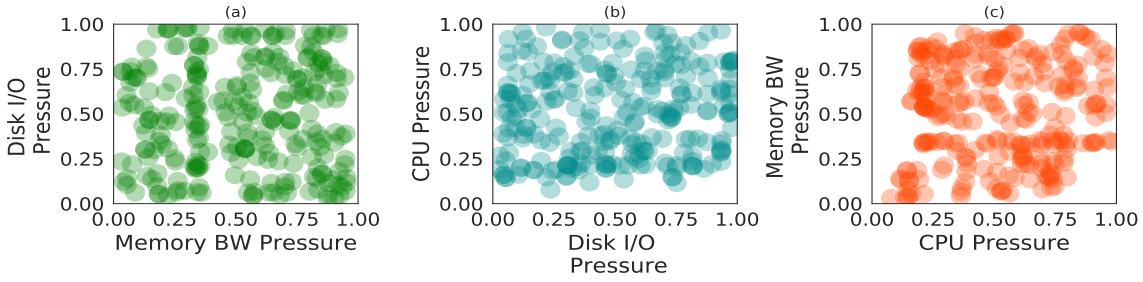

Figure 8: Coverage of the predicted resource stressor design space exerted by the co-located application combinations available in the knowledge base.

II.6.4 Validating the Accuracy of the Performance Models

We used specific applications drawn from the DaCaPo benchmark, the Parsec benchmark and the Keras machine learning application model as target applications whose performance models we were interested in. Specifically, from the DaCaPo benchmark, we chose PMD which is an application that analyzes large-scale Java source code classes for source code problems. From the Parsec benchmark, we chose the Canneal application, which uses a cache-aware simulated annealing approach for routing cost minimization in chip design. The Canneal program has a very high bandwidth requirement and also large working sets [46]. From the Keras machine learning application model, we used InceptionResnetV2, which represents an emerging workload class for prediction inference serving systems [13].

We first build performance models for these three applications using our approach as discussed in Section II.4. To test the effectiveness of the learned application performance models, we co-locate the target applications with the web search workload from the CloudSuite benchmark [76], which uses the Apache Solr search engine framework and emulates varying number of clients that query this web search engine. We ran four different scenarios with varying number of clients: 600, 800, 900, and 1000. We placed our target application with a co-located web-search server on the host compute node. We assigned two cores to the target application and the rest of the cores to the web-search server.

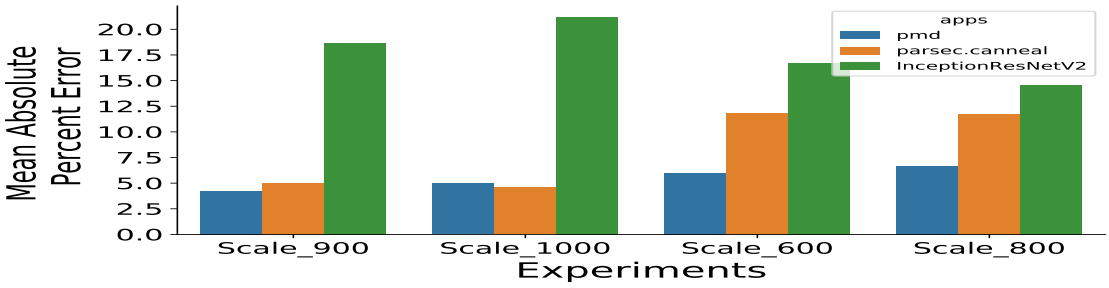

Figure 9: Prediction accuracy in mean absolute percent error for `PMD`, `Canneal`, `InceptionResnetV2` applications when co-located with web search server from CloudSuite.

Figure 9 shows the mean absolute percent errors (MAPEs) for the three applications under varying degrees of loads generated by the clients of the co-located web search application. For example, when the number of clients is 800, the MAPEs are 6.6%, 11.8% and 14.5% for PMD, Canneal and InceptionResnetV2, respectively. Also, the median percentage errors for all the cases are below 5.4%, 7.6% and 13.5% for PMD, Canneal and InceptionResnetV2, respectively. To showcase the total number of correct predictions made by the system, we leverage a CDF curve that has been used in the literature to showcase the effectiveness of the machine learning models [113]. Figure 10 illustrates that, for the PMD application, 80% of the predictions have error rates less than 9%. For the Canneal application, 80% of the predictions have error rates below 15%. For the InceptionResNetV2 application, about 70% of the predictions have error rates below 25%.

II.6.5 FECBench in Action: A Concrete Use Case

Besides validating the efficacy of FECBench on applications drawn from the benchmarking suites, we have applied FECBench to interference-aware load balancing of topics for a publish-process-subscribe system [26]. The Publish/Subscribe (pub/sub) communication pattern allows asynchronous and anonymous exchange of information (topic of interest) between publishers (data producers) and subscribers (data

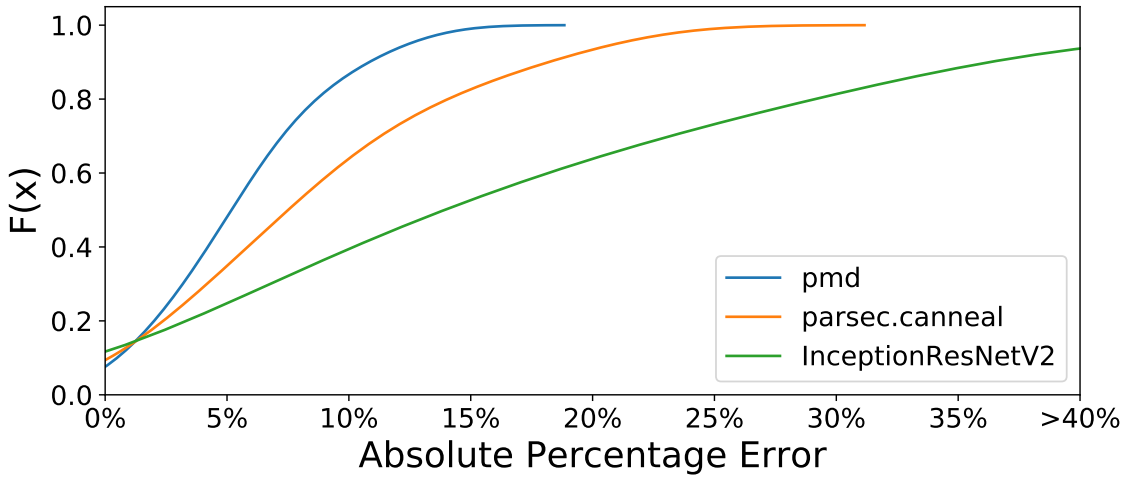

Figure 10: Cumulative distributions of prediction errors for the `PMD`, `Canneal`, `InceptionResnetV2` applications.

receivers). Therefore, pub/sub is widely used to meet the scalable data distribution needs of IoT applications, where large amounts of data produced by sensors are distributed and processed by receivers for closed-loop actuation. The need for processing sensor data is accomplished on broker nodes that route information between publishers and subscribers.

In a publish-process-subscribe system, a topic's latency can suffer significantly due to the processing demands of other co-located topics at the same broker. Figure 11 demonstrates this effect. Here, a topic is characterized by its processing interval p, i.e., average time for processing each incoming message on the topic, and cumulative publishing rate r, at which messages arrive at the topic. Figure 11(a) shows that topics A, B and C show wide variations in their 90th percentile latencies ($\sim$ 100ms to $\sim$ 800ms) under varying background loads.

For latency-critical IoT applications, it is necessary to ensure that a topic's latency is within a desirable QoS value. Therefore, it is important to co-locate topics at the brokers in an interference-aware manner such that none of the topics in the system violate their latency QoS. To this end, one approach is to learn a latency prediction

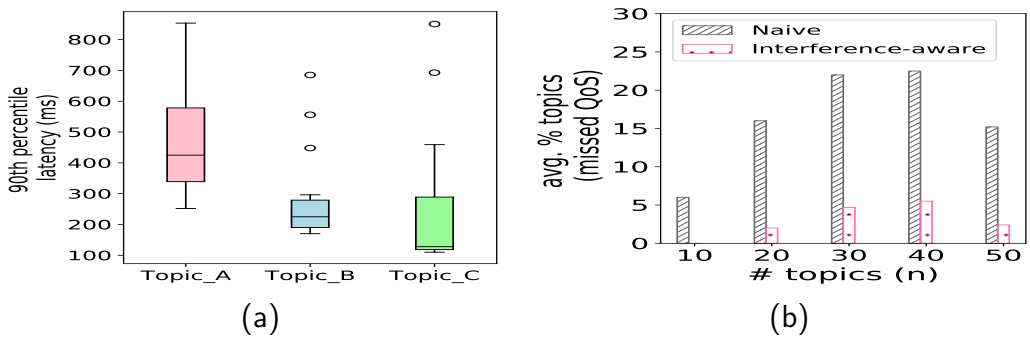

Figure 11: Use of FECBench in publish-process-subscribe showing: (a) impact of co-location on a topic's latency; (b) interference-aware placement of topics.

model for the brokers in the pub/sub system by leveraging the FECBench approach. Subsequently, the latency prediction model can be used to determine which topics can be safely co-located at a broker without incurring QoS violations. Figure 11(b) shows how such an interference-aware method can reduce the percentage of topics in the system that suffer from QoS violations. Here, the interference-aware approach, which uses the latency prediction model obtained by the FECBench approach, is able to meet the QoS for $\sim 95\%$ of the topics in the system. This is significantly better than a naive approach based on round robin scheduling, which is only able to meet the QoS for $\sim 80\%$ of the topics in the system.

II.7 Conclusion

Making effective dynamic resource management decisions to maintain application service level objectives (SLOs) in multi-tenant cloud platforms including the emerging fog/edge environments requires an accurate understanding of the application performance in the presence of different levels of performance interference that an application is likely to encounter when co-located with other workloads. Data-driven performance models can capture these properties, which in turn can be used in a feedback loop to make effective resource management decisions [43, 48]. The vast number of applications, their co-location patterns, differences in their workload types, platform

heterogeneity and an overall lack of a systematic performance model building framework make it an extremely daunting task for developers to build such performance models. FECBench (Fog/Edge/Cloud Benchmarking) is a framework that addresses these challenges, thereby relieving the developers from expending significant time and effort, and incurring prohibitive costs in this process. It provides an extensible resource monitoring and metrics collection capability, a collection of disparate benchmarks integrated within a single framework, and a systematic and scalable model building process with an extensible knowledge base application combinations that create resource stress across the multi-dimensional resources design space. Empirical evaluations on different application use cases demonstrate that the predicted application performance using the FECBench approach incurs a median error of only 7.6% across all test cases, with 5.4% in the best case and 13.5% in the worst case. FECBench is available in open source at `https://github.com/doc-vu/fecbench`.

So far, FECBench has been evaluated on a single hardware platform. Its efficacy needs to be validated on a variety of hardware platforms. To that end, we will explore the use of transfer learning to minimize the efforts. Our use of 106 applications did not provide coverage across every possible resource dimension and hence improving the coverage is another area of future work. As a part of research outreach of the FECBench, the students from the "Visual Analytics & Machine Learning CS8395.03" course at Vanderbilt University during Spring 2019, explored various visualization techniques for better visual exploration of the performance effects seen in the applications running on the multi-tenant servers.

CHAPTER III

ADDRESSING ACCESIBILITY CONCERNS IN PERFORMANCE MODELING

III.1 Introduction

Multi-tenancy in cloud deployment and changing workload patterns for cloud-based applications make it hard to analyze and diagnose any incurred performance issues and find appropriate solutions to resolve them. In particular, identifying the sources of performance interference due to multi-tenancy remains a hard problem [31, 67, 136].

To support our hypothesis, consider Figure 12, which shows the performance variabilities incurred by a multi-tenant cloud deployment of an image recognition application based on Inception-Resnet v2 Keras machine learning model [14]. As seen, the performance deteriorates significantly when running in a multi-tenant environment compared to a baseline performance with no resource contention. Understanding these performance issues, which themself can change dynamically, is important to make effective dynamic deployment and resource management decisions so as to meet applications' service level objectives (SLOs).

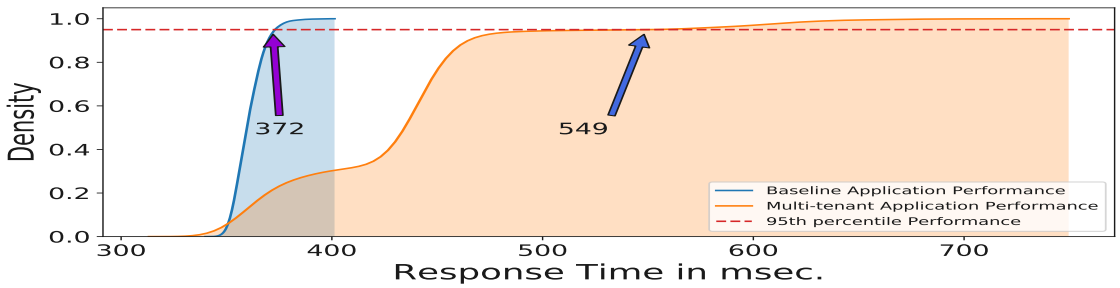

Figure 12: CDF of Response Time for Inception-ResNet v2 model using Keras with 4 Cores

A number of resource- and application performance-monitoring frameworks have been developed in recent years, e.g., Nagios [27], Zabbix [122], Intel SNAP, linux-perf, collectd [78] and systat to name a few. These performance monitoring tools provide the users with different system-level and in some cases application-level metrics, which in turn provide insights into the runtime performance of these applications.

As is the case with any technology, using such tools often involves a steep learning curve in understanding their usage (e.g., their APIs), their features and capabilities, and often requires users to manually write custom configuration scripts and programs to effectively utilize the tools. With advances in hardware that enable more finer-grained performance metrics to be collected, these problems are further excarbated. For example, Intel has recently introduced the Cache Monitoring Technology tools which are compatible with the new generation of Intel architectures [4]. To use these capabilities, new monitoring tools and programs need to be added to capture the desired performance statistics. Further, a user must possess expert knowledge about the hardware architecture and monitoring tools.

Recent efforts [25, 84, 110, 149] have attempted to address these concerns. However, there still remain many unresolved issues that must be addressed. For instance, such solutions are usually tightly coupled to an execution platform and as such do not support compatibility with newer platforms. Secondly, many tools are non-intuitive to use and less user friendly thereby requiring users to manually configure and install the monitoring probes on the runtime platforms, which is an error-prone process. Beyond addressing these limitations, newer capabilities are needed to enable runtime performance monitoring of these applications. For instance, older frameworks often cannot be extended to exploit finer-grained or newer hardware metrics, such as last-level cache utilization or non uniform memory access (NUMA) patterns.

To overcome the above challenges, we propose to utilize the principles of software product lines (SPLs) [62] in creating a model-driven generative framework called

UPSARA- ***U****nderstanding* ***P****erformance of* ***S****oftware* ***A****pplications and* ***R****untime System* ***A****nalysis*. A product line is essentially a family of product variants which have a set of features common to all variants, but differ from each other along some other features of an overall feature set that defines a product line. Our approach is based on the observation that different monitoring frameworks share many common features, while at the same time differ on some other key features. Thus, the different monitoring frameworks can be seen as variants of a product line. Using generative capabilities provided by model driven engineering (MDE) [134], we synthesize the configurations for the different frameworks and automate the desired performance monitoring tasks for the use case under consideration.

The rest of the chapter is organized as follows: Section III.2 presents the motivation, requirements and the architecture of UPSARA. Section III.3 delves into the details of the model driven engineering techniques, and the UPSARA domain specific modeling language. Section III.4 describes the generative capabilities of UPSARA. We present validation of UPSARA using representative usecases in Section III.6. Related work is described in Section III.7. Finally, we present concluding remarks and future directions for UPSARA in Section III.8.

III.2 Design and Implementation of UPSARA

In this section we highlight the key challenges and solution requirements, followed by an overview of the UPSARA solution.

III.2.1 Eliciting Challenges and Solution Needs

The stakeholders of UPSARA are cloud performance engineers who would like to analyze an application's performance on the target platform. An application can be a monolithic application such as a database, a micro-service based component assembly, or can be a distributed application such as Map-Reduce, distributed co-simulations,

or parallel processing jobs. To analyze the performance delivered to an application, an engineer needs to collect system metrics such as the CPU, network, disk, memory utilization as well as micro-architectural metrics, such as context switches, cache utilization, memory interconnect utilization, among others. These metrics are needed to pinpoint performance interference issues incurred due to multi tenancy. Moreover, the engineer will also need application-level performance metrics, such as observed response times and throughput.

The system metrics collection is typically performed using external programs such as collectd, statsD, likwid, linux-perf, Intel PMU tools, Intel RDT tools, among others [75]. Such tools can measure some or all of the metrics of interest. As such, one may need a single or a collection of such tools to cover the spectrum of metrics of interest. The runtime platform and the application then needs to be configured accordingly with the right collection of tools based on the metrics selected by the performance engineers. Each tool may impose different approaches for its configuration such as through *.conf* files, or by passing input parameters during program invocation. The target platform also might have limitations as to which metrics it can offer to be monitored, and what tools need to be executed to capture the supported metrics.

Based on these concerns, below we describe the key requirements for a solution like UPSARA that we present in this chapter.

1. ***Ease of Use***: The metric instrumentation framework should have a lower entry barrier so that it is easy to use in the continuous integration and performance studies of a cloud-hosted application. The steep learning curve for individual tools hinders the users from making use of the features provided by such platforms. Thus, UPSARA should provide intuitive and higher-level abstractions, which hide the lower-level complexity thereby making it more easy for the end users to utilize the platform.

2. ***Ensuring the correctness of the configuration:*** It is important that the metrics selection on a platform are supported by that platform. For instance, monitoring *non-uniform memory access* (NUMA)-level statistics or measuring cache statistics requires that the underlying platform hardware support these capabilities. Without such support, forcing the monitoring tool to capture these parameters will either result in capturing garbage data or throw a run-time exception complaining about the non-existence of such features. Hence, UPSARA should natively support built-in correctness or a violation checker, which will enforce *correct by construction* design.

3. ***Well-formedness of generated artifacts:*** To avoid writing low-level code artifacts, generative programming has helped developers by synthesizing various code artifacts based on user-defined templates. Although, generative programming can synthesize these artifacts, it is essential that a generative solution adopted by UPSARA be correct and adhere to the domain-specific rules. This is necessary to ensure functional correctness of the system.

4. ***Support for heterogenous runtime architectures***: UPSARA should be able to support heterogeneous runtime architectures, which are common in cloud environments. Since each such runtime architectures might have their own set of metric monitoring tools, UPSARA should support seamless and automated composition of such tools in its architecture.

5. ***Extensibility and tool reuse:*** UPSARA must be able to reuse the existing capabilities of underlying measurement tools that provide monitoring of system metrics rather than reinventing the wheel by developing new monitoring tools. The framework should support semantics for easy plug and play architecture for adding support for new tools. Also, as new hardware architectures get

developed, UPSARA should be able to add support for new architecture metric measurements.

III.2.2 Architecture and Workflow

Figure 13 illustrates the high-level operational workflow of UPSARA that performance engineers can use for analyzing their cloud-hosted applications. To begin with, the designer of the experiment provides a high-level specification of the performance analysis to be conducted. The specification includes the various metrics to be measured, the type of application to be studied, and information about the runtime platform on which the application needs to be executed. The designer encodes this specification using the visual elements of UPSARA's domain-specific modeling language (DSML) (See Section III.3). Once the specification is captured, UPSARA transforms the specification into valid configuration scripts and deploys the artifacts onto the runtime platform. Application execution then begins on the target runtime platform. The system metrics and the application metrics are captured during the execution of the application and are made available to the user by means of auto-analysis and visualization charts.

We now describe the main building blocks of UPSARA.

- *WebGME Visual Environment:* This block provides an intuitive interface via the DSML for designing and orchestrating application performance analysis experiments. The DSML-based interface enables (1) configuring various system metrics to be collected on the target runtime platform, (2) controlling the execution of the experiments, and (3) providing analysis and real-time system dynamics.

- *Model Interpreter:* The model interpreter validates the well-formedness of the

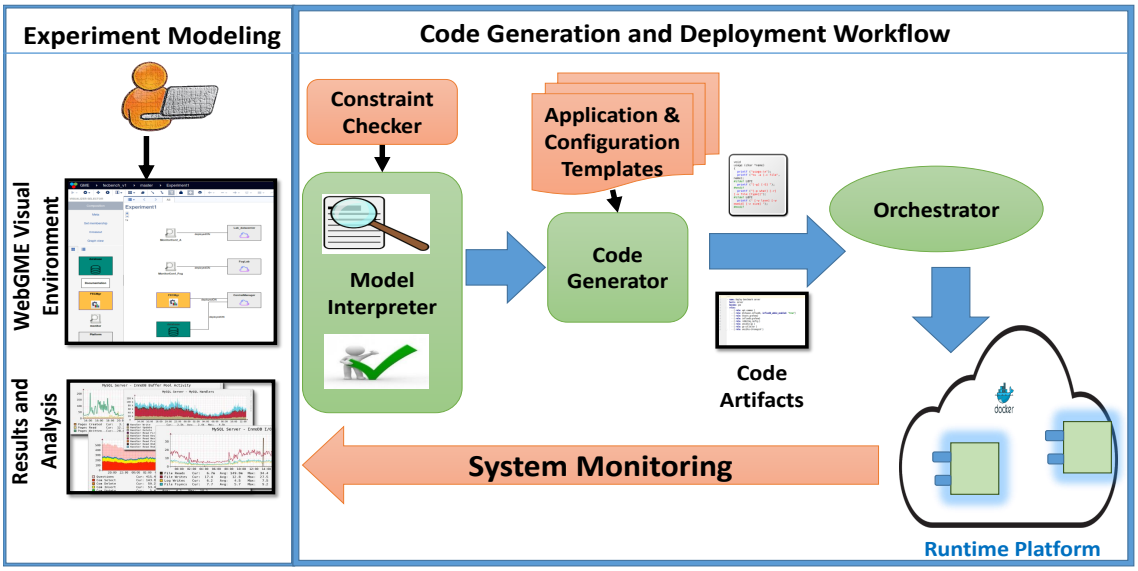

Figure 13: High Level Overview of UPSARA

user-supplied model by traversing the model elements and validating their syntactic and semantic correctness while also generating the desired artifacts. The generative aspects of the interpreter handled by the Code Generator block (described below) implements a graph traversal logic using a visitor design pattern [55] and synthesizes artifacts that capture the relationships and model attributes as described by the user in lower-level representation.

- *Constraint Checker:* This module is responsible for detecting any violation in the design of the experiment. It consists of rules which specify the correct properties of the application execution. Examples of constraints are *an application can be deployed only on one runtime platform at a time* or *supported list of metrics of the runtime platform.* Thus, if the designer of the experiment constructs a scenario where the application is deployed on two different platforms at once, a constraint violation action will be triggered notifying the user of the violations. These constraints are captured and are made available in the rules database of UPSARA.

- *Code Generator:* This component is responsible for generating the desired artifacts as specified by the user-supplied model. The code generator has access to a repository of application and configuration template schemas as required by the underlying monitoring and deployment tools. As an example, the generator can synthesize the schemas for the metric monitoring software. Also, based on the application deployment on the target runtime system, the code generator can generate scripts called *playbooks* that will then be input to an application orchestration framework called *Ansible* [3]. The code generator is also responsible for generating visualization artifacts for viewing the results.

- *Orchestrator:* This component interfaces with the runtime platform. The orchestrator is responsible for deploying the generated artifacts on the target platform by instantiating appropriate application deployment and metrics collection.

- *Result and Analysis:* This component is responsible for presenting the user with the analysis and statistics of the application performance as designed by the user. It features Python notebooks hosted on a Jupyter server [97]. The graphical visualization allows users to analyze various application performance characteristics. These include the resource consumption for different system metrics such as CPU, load, memory, energy, etc. Specialized analyses modules can be integrated into the generated notebooks which provide relevant analysis for the selected frameworks.

III.3 UPSARA's Domain-Specific Modeling Language (DSML) Design

We now delve into the details of UPSARA's domain-specific modeling language (DSML).

III.3.1 Encoding UPSARA's Product-line Feature Model

We leverage model-driven engineering (MDE) techniques to encode the different elements involved in UPSARA's product line for aiding in performance analysis of cloud applications. Specifically, the feature model is encoded as a meta-model(s) of an underlying DSML. To design the DSML, we have used the WebGME modeling environment [106] to create the meta-models, writing model interpreters, and encoding generative logic for synthesizing artifacts. WebGME itself is a cloud-based service that provides a browser-based design environment and supports creating visual DSMLs.[1]

The WebGME environment includes tools to write the model interpreters using the default *NodeJs* language. It also has support for writing model interpreters in other programming languages such as Python and Java. WebGME supports capturing DSML syntax and its semantics via meta-modeling. Model interpreters can be programmed for each specific meta-model instance. Model-interpreters also provide an additional avenue for capturing information, which otherwise cannot be explicitly captured in the meta-model. For example, capturing constraint violation logic can be encoded in the model-interpreters.

UPSARA's visual DSML provides several benefits when compared to manually written scripts. The visual component blocks provide an easy way to instrument and construct an experiment which can then be deployed as specified [34, 35, 42]. This reduces barrier to entry for developers that can rapidly and concisely describe and start running experimental scenarios without learning a new programming language terminology and syntax [61]. The visual blocks are intuitive and hence the learning curve for using UPSARA is negligible. Next, we describe the main aspects of the UPSARA DSML and their responsibilities.

[1] A textual DSML is an alternative approach but we did not explore it.

III.3.1.1 UPSARA Main Meta-model

Figure 14 shows the main building block of the DSML. It includes the following meta-model components:

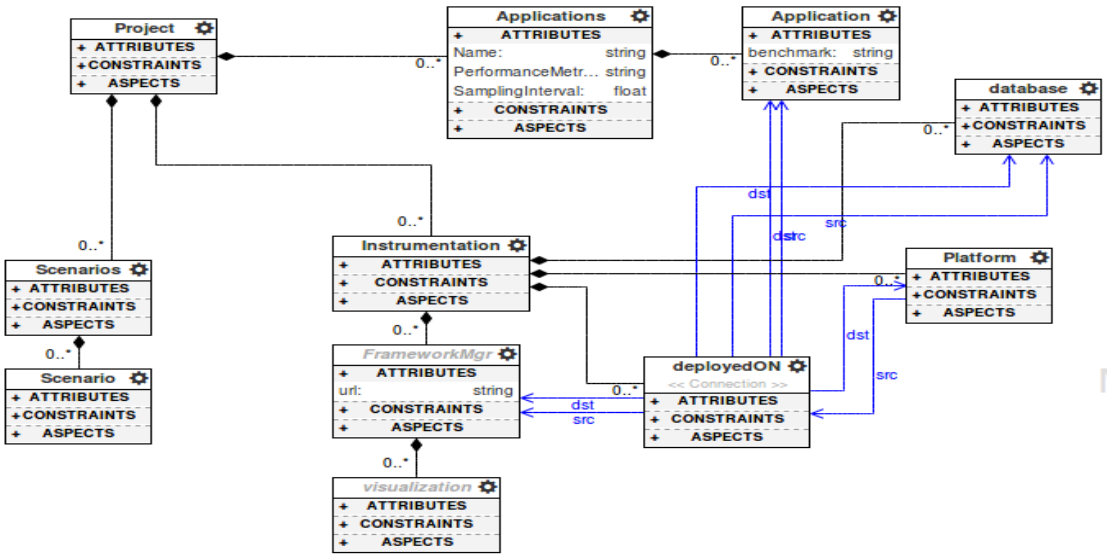

Figure 14: UPSARA Main Meta-model

Project: represents the top-level meta-model block of UPSARA. *Project* comprises an *application* set, and includes information about the *instrumentation* metrics to be collected on the runtime *platform*. The *Project* component also contains a *scenarios* model. *Scenarios* describe different deployment schemes for running applications on the runtime platform.

Platform: The platform represents the cloud platform on which the application performance study will be conducted. This usually represents target host machines which includes: virtual machine, lightweight containers (e.g., Docker), bare-metal, as well as a cluster of host machines.

FrameworkMgr: This component represents the monitoring framework which

the user can configure with the desired set of metrics to monitor. Individual frameworks can be custom-built with a predefined set of metrics that are supported for the specific performance monitoring activity. Specialized frameworks can be built by extending the *FrameworkMgr* to suit different application performance monitoring scenarios.

***Instrumentation**:* This component defines how the metrics collection will be configured on the underlying cloud platform using one or more concrete frameworks, e.g., collectd. The metrics are associated with *frameworkmgr*. Moreover, the runtime platform is represented by *platform*. The relationship between *frameworkmgr* and the *platform* is represented by the connection semantics in WebGME.

deployedON: This represents connection semantics in WebGME which capture the relation between two nodes. Here *deployedON* represents a connection between *FrameworkMgr* and *Platform*, *database* element and *platform*.

Scenarios: As mentioned earlier, this component includes the different deployment options for running the applications on cloud platforms. A user may want to study application performance on multiple runtime platforms at once. For such usecases, the user will define a scenario that captures the *application to the runtime platform* mapping. *Scenario* component facilitates describing such mapping for the performance study.

***Visualization**:* Visualization of data and results is provided as an important aid in conducting performance analysis of applications. The *visualization* component generates plots that are specific to the *FrameworkMgr* being developed. Every *FrameworkMgr* instance can have a custom set of visualization artifacts that meaningfully represent the performance analysis data for the framework. Visualization can be produced using a set of Jupyter notebooks [97] that capture specifics of the application performance.

Database: This component is used for storing the collected performance metrics data.

III.3.1.2 UPSARA Metric Meta-model

Figure 15 showcases a snippet of the metric monitoring meta-model. The system metric to be monitored can be macro-level metrics such as CPU, memory, network, disk, load, as well as micro-architectural metrics such as the context switches, L1 cache, L2 cache, L3 or LLC cache utilization, cache hit and miss counts, scheduler specific metrics, Docker container-specific metrics, etc. Figure 15 shows a snippet of the micro-architectural meta-model. As shown in Figure 15, using attribute selection we allow fine-grained access to specific aspects of the system metrics that a user would want to collect. As an example, the figure shows that cache-miss property can be set to *TRUE* or *FALSE*.

III.3.2 UPSARA Platform Meta-model

Figure 16 illustrates the cloud platform meta-model. *Platform* represents the target runtime platform on which the application performance needs to be studied. *Platform* could be any of baremetal host machine (*Host*), a virtual machine (*VM*), or a lightweight Docker container (*Docker*). All these are derived from an abstract *machine* concept in the DSML. *Cluster* comprises a group of *machine* elements. A *cluster* could be either in the form of a *Datacenter*, *FogPlatform*, or an *EdgePlatform* that represents the clouds, fog and edge computing resources, respectively [49].

III.3.2.1 UPSARA Application Deployment Scenario Meta-model

To test the performance of an application on various platforms we provide an application deployment *scenario* meta-model. The scenario meta-model shown in Figure 17 illustrates how an *application* maps to runtime platforms.

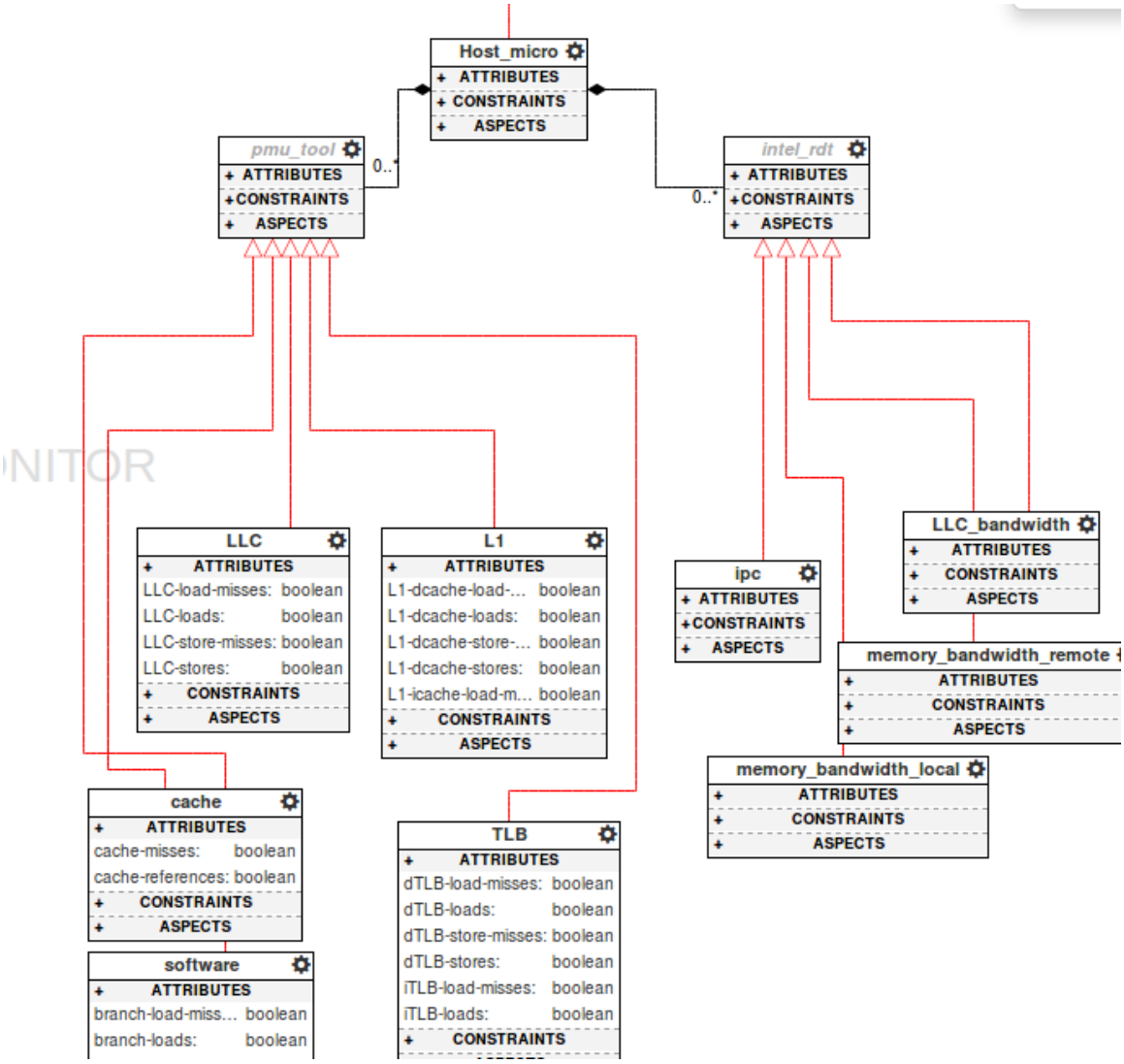

Figure 15: Snippet of UPSARA Micro-metric Meta-model

A scenario for an experimentation can involve an application or a group of applications being deployed on a single or a set of runtime platforms. *ApplicationRef* references the *Applications* meta-model. Similarly, *MachineRef* references the *machine* meta-model. As an example, a user might be interested in knowing how an application runs on platform A, and would also like to see the application's performance on platform B. Using the UPSARA language one can easily create these application deployment scenarios by reusing existing artifacts and simply referencing them.

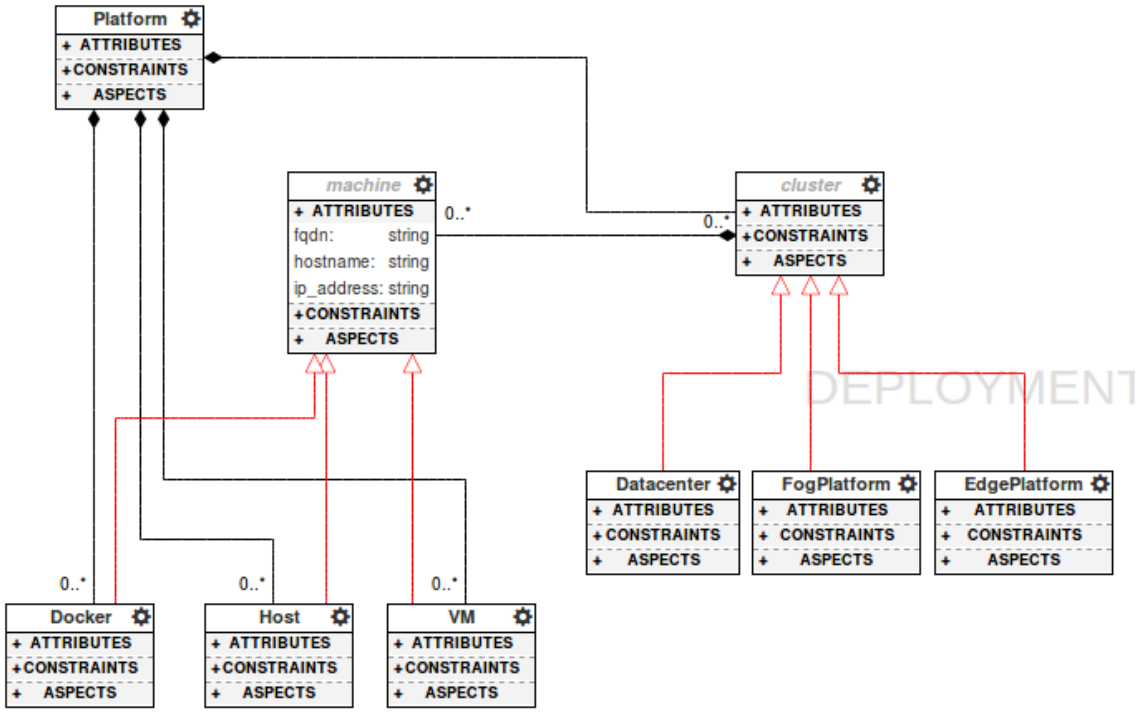

Figure 16: Snippet of UPSARA Runtime Platform Meta-model

III.3.2.2 UPSARA Specialized Framework Meta-model

To cater to the commonalities and variabilities of the concrete metrics collection frameworks, we provide extensible meta-models with constraints for each such framework that can be specialized from the abstract meta-model. An example of a specialized framework that can be built using the UPSARA DSML is shown in Figure 18. This specialized framework, which we have named as *FECBench*, has been configured to support a subset of monitoring metrics. These include *cpu_util*, *Docker_macro*, *Docker_micro*, *memory_bandwidth_local*, *memory_bandwidth_remote*, and *LLC_bandwidth* system resource metrics. Also, the *fec_viz* object of the type *visualization* meta element is included as a part of this *FECBench*. *fec_viz* includes information about the custom type of visualization that is needed to support analysis for the *FECBench* framework.

This example shows how users create a concrete product line variant from the

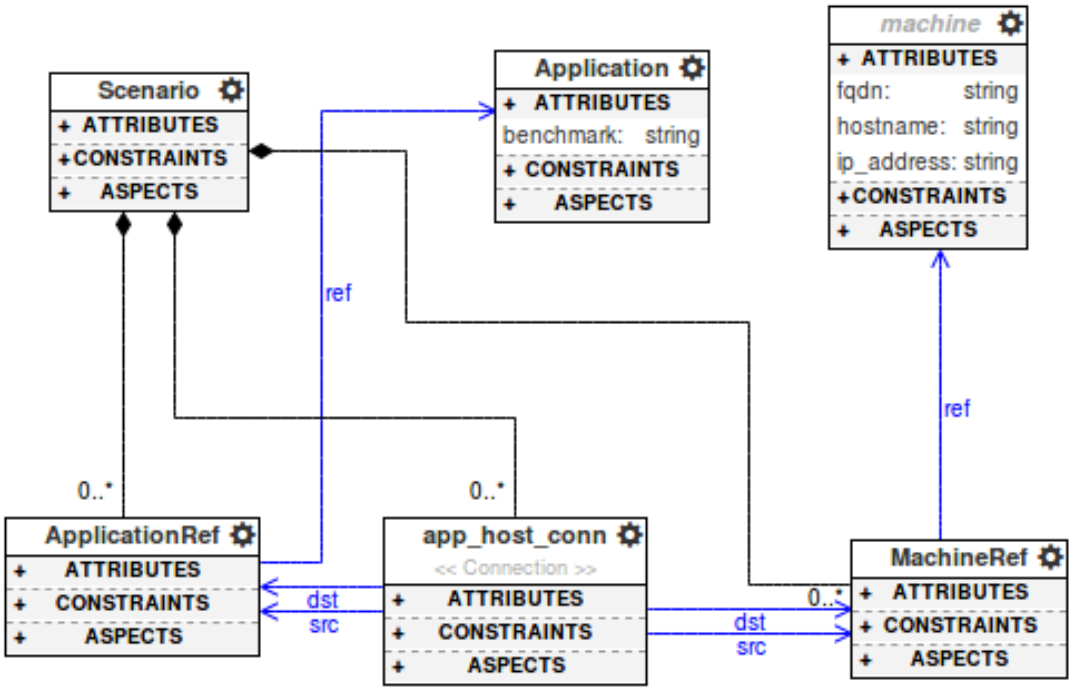

Figure 17: Snippet of UPSARA Scenario Meta-model

UPSARA DSML and bring more specialized visualization and analysis aspects as required for each of the custom frameworks. Also, using this meta-model design, the framework can enforce design constraints. For example, it only allows the subset of the metrics as defined by the specialized framework to be available to the end user thereby enforcing constraints and following the *correct-by-construction* principle [134].

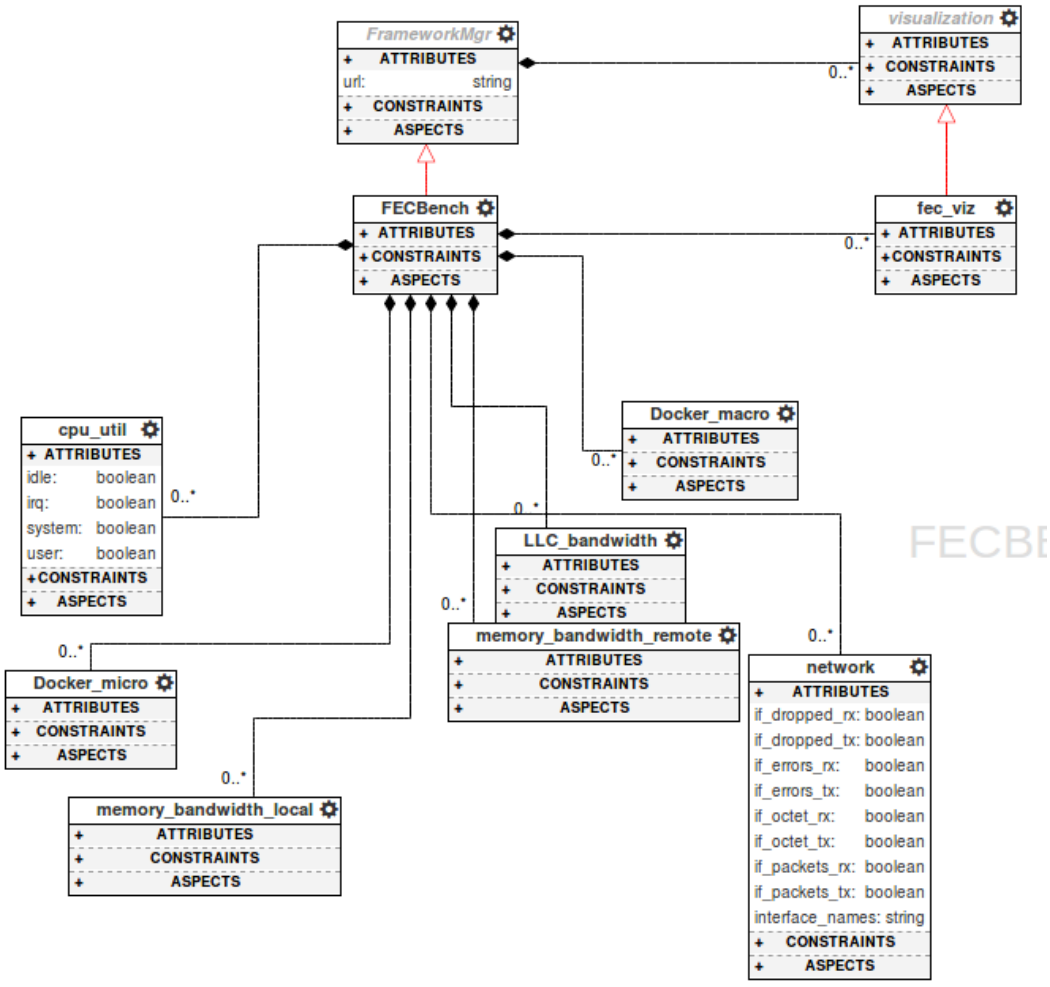

Figure 18: UPSARA Specialized Framework Meta-model

III.4 UPSARA's Generative Capabilities

UPSARA's generative capabilities are built using the WebGME tool. In WebGME terminology, the model interpreters and the generation can both be handled by writing custom programs called *plugins*. Thus, UPSARA implements the model interpreters and the generators using the *plugins* infrastructure. The *plugins* can be instantiated by users or they can run as service processes. In UPSARA, instantiation of plugins happen when users invoke the plugin by clicking the plugin button in the WebGME

environment. Figure 19 shows the process of generation, synthesis and deployment in the UPSARA's generative toolchain.

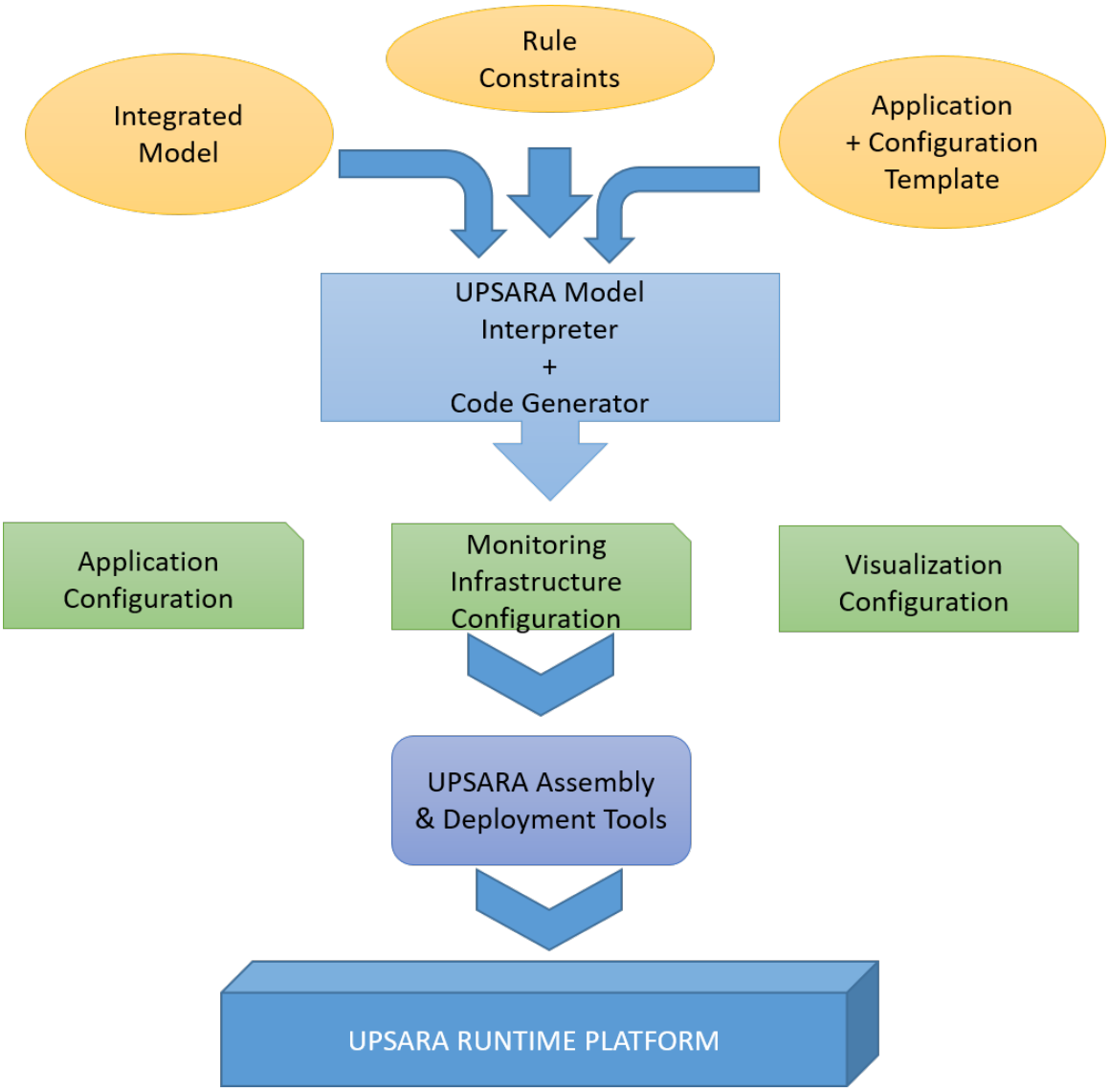

Figure 19: UPSARA Generative Process

We leverage the *Embedded Javascript*(EJS) [8] templates for creating templates of the code artifacts that needs to be synthesized. EJS templates also supports encoding custom logic which can be utilized for implementing constraints checking logic.

III.4.1 Metric Monitoring Configuration Generation and Provisioning

As described in Section III.3.1.2, a range of metrics collection can be modeled in UPSARA. However, based on the actual underlying metrics collection framework used, only a subset of the metrics are realistically available to the end user at runtime. An example of such a specialized framework is illustrated in Figure 18. Also as shown in Figure 14, there is a mapping between the specialized framework and the platform on which it needs to be deployed. Each metric that needs to be measured can be configured to be monitored from a set of available system monitoring tools. The generative capabilities of UPSARA capture the above relationships between different entities. Moreover, based on the target monitoring tools, UPSARA synthesizes appropriate configuration scripts.

Algorithm 1 depicts the steps involved in the metric configuration generation and deployment. As shown, the generator first needs to traverse the model and get a list of metrics and the hosts on which the metrics are to be configured (Line 2, 3). UPSARA then checks if the selected metric is actually supported by the target runtime platform. This platform capability information is pre-populated and is available to the generator for lookup. If the metric is not supported by the underlying platform, a constraint violation is registered as shown in Line 8. If the metric is supported, we first get the monitoring tool information associated with the metric (Line 9).

Once the appropriate monitoring tool is determined, the associated tool's configuration template is fetched. UPSARA generates an *Ansible* playbook for configuring that metric on the selected cloud platform. UPSARA also generates the configuration script for the monitoring tool to configure the metric selection.

Moreover, as a part of monitoring and analysis, a Jupyter notebook template associated with the framework is loaded and deployed as shown in Line 20. The jupyter notebook files have `.ipynb` extension and comprise a list of `json` objects.

Algorithm 1: Metric and Visualization Configuration Generation and Deployment

Input : Model

```
1  ListofMetrics ← getMetrics(Model)
2  ListofHosts ← getPlatformNodes(Model)
3  iframeworkViz ← getFrameworkViz(Model)
4  mapMetricToHost ← getMappingofMetricsToHost(Model)
5  if mapMetricToHost! = ∅ then
6      for V_{m,h} ∈ mapMetricToHost do
7          if isMetricSupportedbyHost(m, h) = False then
8              Skip ; // Constraint Violation
9          metricTool ← getMetricTool(Model, m)
10         metricConfTemplate ← getTemplate(m, metricTool) ; // Generate
               Artifact
11         hostFiles[h].insert(metricConfTemplate)
12     end
13     for host ∈ hostFiles do
14         for files ∈ hostFiles[host] do
15             InstantiateDeployment(host, files) ; // Deploy Artifact
16         end
17     end
18 if iframeworkViz! = ∅ then
19     vizTemplate ← getVizTemplate(iframeworkViz)
20     deployTemplate(vizTemplate) ; // Deploy Visualization
21 end
```

The jupyter notebook can be dynamically populated with the metrics parameters that need to be monitored whose information is available in the model.

III.4.2 Application Configuration Generation and Orchestration

The process for obtaining application-specific details is similar to the scheme described in Algorithm 1. The generator program first loads the model and gets the information about the applications to be studied. Next it finds the association between the application and the deployment platform on which it needs to be studied. Once the target applications and the runtime platform are determined, UPSARA generates a configuration script. This configuration script is further used by the main framework program which performs the application deployment and execution on the target platform.

III.5 Meeting The Requirements

Based on the description of UPSARA's DSML design and generative capabilities, we now show how UPSARA addresses the challenges and meets the requirements described in Section III.2.1.

1. Ease of Use: UPSARA provides a visual DSML for easy construction of very large scale experimentation. The performance engineer can also easily and rapidly select metrics of interest which need to be analyzed for the application under test. The generative capabilities in UPSARA create monitoring metric configuration scripts without the need for manual writing of scripts for configuring the monitoring tool.

2. Ensuring the correctness of the configuration: The UPSARA DSML embeds the relationship and constraints between the meta elements of the language. As such these rules ensure that when the user starts designing experiments using WebGME, only the constructs that are supported by the language are available and

visible to the user for instantiating. Also, the model interpreter enforces constraint checking which ensures that the experiment that is designed by the user also is checked for constraint violations. This is addressed in Section III.4.1.

3. Well-formedness of the generated artifacts: UPSARA provides automated synthesis of application configurations, experimental configurations, metric configurations and the result visualization artifacts. UPSARA's generative process as described in Section III.4 includes constraint checking rules that detect any violations encountered in the model design. Capturing the violation allows it to ensure that the artifacts generated actually ensure correct functional aspects of the performance measurement study.

4. Extensibilty and tool reuse: Section III.3.2.2 explains how a user can create a specialized framework using existing elements from the UPSARA DSML. It also demonstrated how a user can use existing language elements and build new models using the UPSARA DSML.

5. Support for heterogenous runtime platforms: Section III.3.2 discusses how the UPSARA platform meta-model is able to support heterogeneous runtime platforms. The correct-by-construction generative and deployment capabilities eases the generation and deployment of large-scale configuration artifacts on the heterogeneous runtime platform. This also avoids the accidental complexities involved in configuring such large-scale systems.

III.6 Evaluating UPSARA via Use Cases

In this section we use three different use cases and their analyzed performance data to highlight the significant benefits derived from using UPSARA. For our use cases, we have used an experimental setup comprising an Intel Xeon processor whose configuration is listed in Table 2. The software details utilized are as follows: the underlying metrics collection frameworks are Collectd (v5.8.0.357.gd77088d), Linux

Perf (v4.10.17) and Likwid Perf (v4.3.0). Configuring different monitoring metrics is achieved by utilizing UPSARA. We used applications from the PARSEC [46], SPLASH-2 [46], and DaCaPo [47] benchmarks for the performance analysis.

Table 2: Hardware & Software Specification of Compute Server

Model Name	Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz
Number of CPU cores	16
Memory	32 GB
Operating System	Ubuntu 16.04.3 64-bit

III.6.1 Case Study 1- Co-located Workload Performance Analysis

Recent literature indicates that datacenter resources often go underutilized [67]. To increase the utilization, cloud providers tend to consolidate applications by means of overbooking. However, co-location of applications without proper knowledge of the applications' performance profile can result in performance interference, which degrades performance. This occurs due to contention for shared resources. In this use case, we use UPSARA to measure the application's degradation with co-located background workloads.

Our use case application is a computer vision application performing image feature recognition. The image processing application uses Scale Invariant Feature Transform (SIFT) to find the scale and rotation independent features of an image [140]. The application is a server-side application with the clients continuously sending an image of fixed size to be processed. We measure the server-side execution time for processing a single client request. The image processing application is also co-located with other background workloads that share resources on the same host machine.

Figure 20 depicts that the performance of the image processing application deteriorates significantly when co-located with background loads due to varying interference impact. To analyze this variation in the performance of the application, the user utilizes UPSARA to configure the metrics to be monitored, and also to execute the image processing client and server application. Using UPSARA, the user leverages UPSARA's runtime machine learning approaches to determine the dominant metrics that are highly correlated with the application's QoS metric: *execution time*. The figure reveals that L1-iCache-Loads, IPC, IPS, L1-dCache-Loads, LLC-loads, cache-miss are highly correlated with the performance of the application. Using this knowledge, intelligent resource schedulers can be designed, which avoid placing this application with other co-located workloads that are not compute and cache intensive.

III.6.2 Case Study 2: Application Resource Utilization Modeling

When deploying applications in the cloud environment, resource management solutions need to allocate sufficient resources to meet application SLOs. The rapid growth of new type workloads such as deep learning and distributed machine learning, are moving to the cloud. As such, studying how different resources are utilized by such workloads becomes critical for cloud/cluster schedulers. As an exemplar, we use the same benchmarks as before for analysis. Our aim was to study the resource utilization of different applications from these benchmarks. Figures 21 and 22 depict the normalized number of system context switches and shared memory bandwidth utilization obtained by orchestrating monitoring components using the UPSARA framework. Due to space constraints we are not displaying rest of the resource utilization metrics. This collected information on resource utilization can be leveraged in building predictive resource utilization models for co-located workloads. Such predictive models can be useful for cloud providers in improving resource allocation algorithms.

III.6.3 Case Study 3: Studying the Impact of Application's Resource Configuration

As more and more jobs are migrated to the public cloud, users are faced with a challenge of which configurations to select from a range of virtual machine options provided. The cloud users usually would like to find a cost effective configuration which meets their application's QoS metric. We use UPSARA to measure the application's QoS metric - *execution time*, for different application container configurations. We measure the execution time for five different CPU core resource configuration options: [1,2,4,8,16] cores. Figure 23 shows the impact of resource configuration on the execution completion times for applications. Using these insights appropriate configuration can be selected such that it satisfies the user requirements.

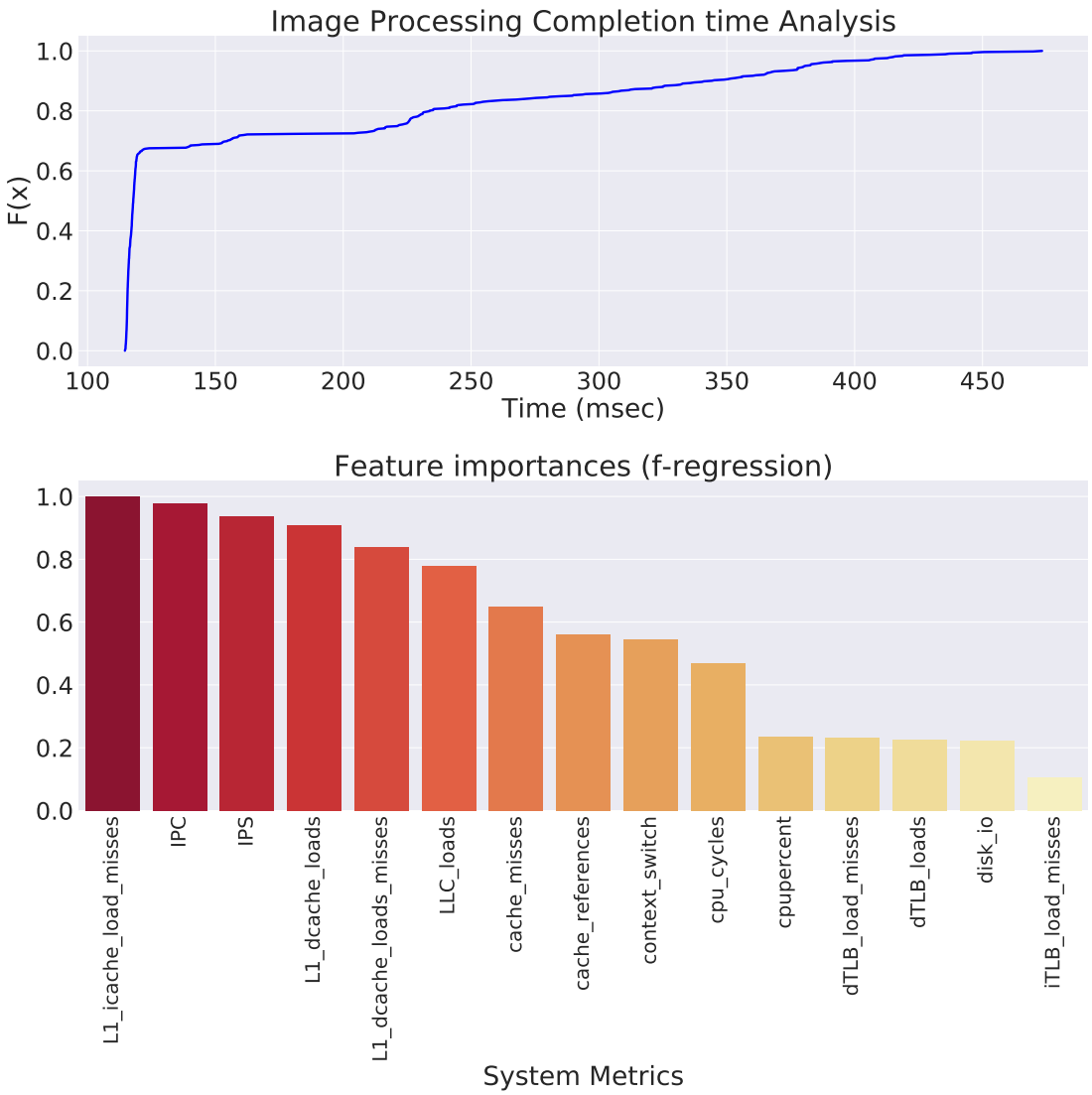

Figure 20: Performance analysis of image processing service. a) Shows the latency distribution of the completion times of the image processing. b) Shows the dominant measurement metrics that are correlated with the increase in the latency of image processing

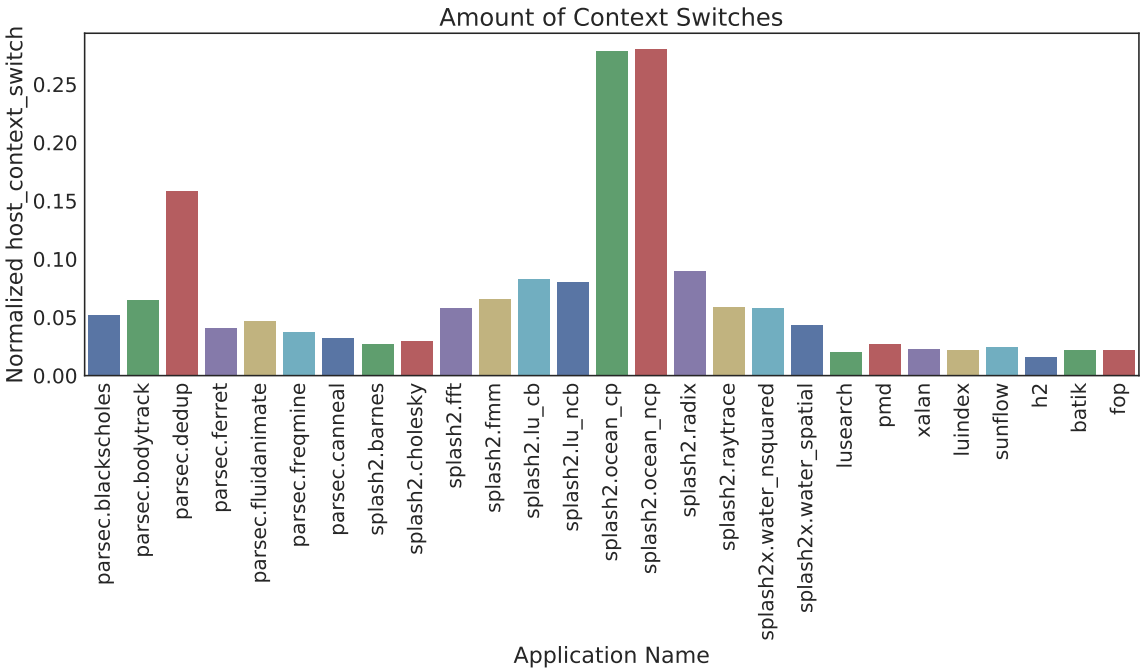

Figure 21: Normalized amount of context switches imposed by applications from the PARSEC, DaCaPo and Splash-2 benchmarks.

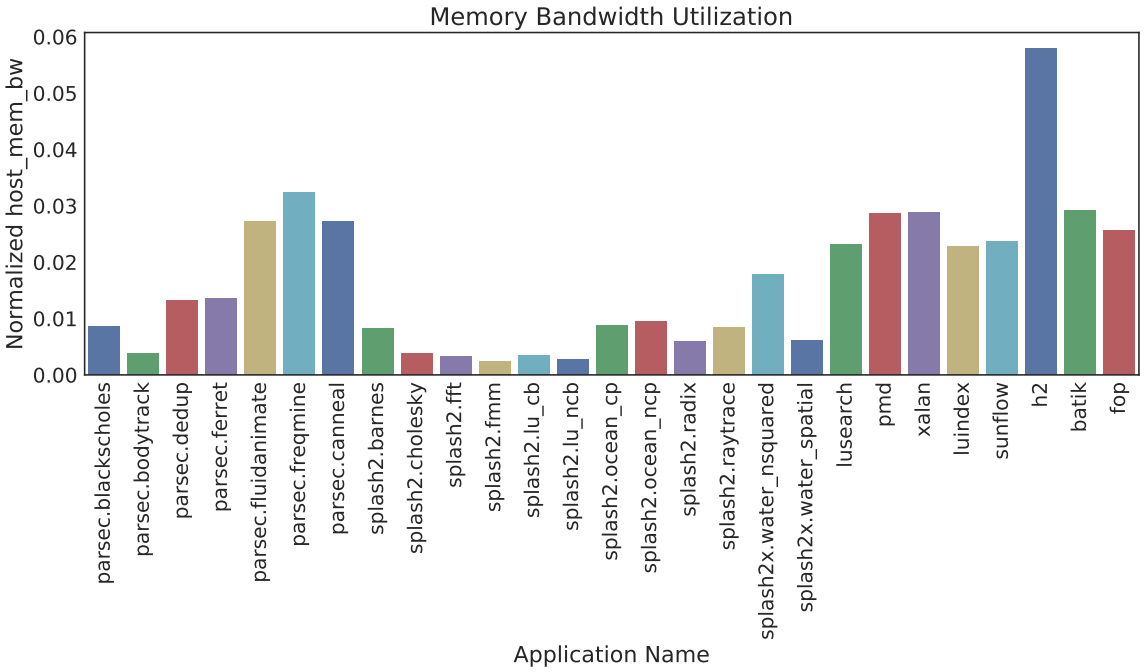

Figure 22: Normalized memory bandwidth utilization of applications from the PARSEC, DaCaPo and Splash-2 benchmarks.

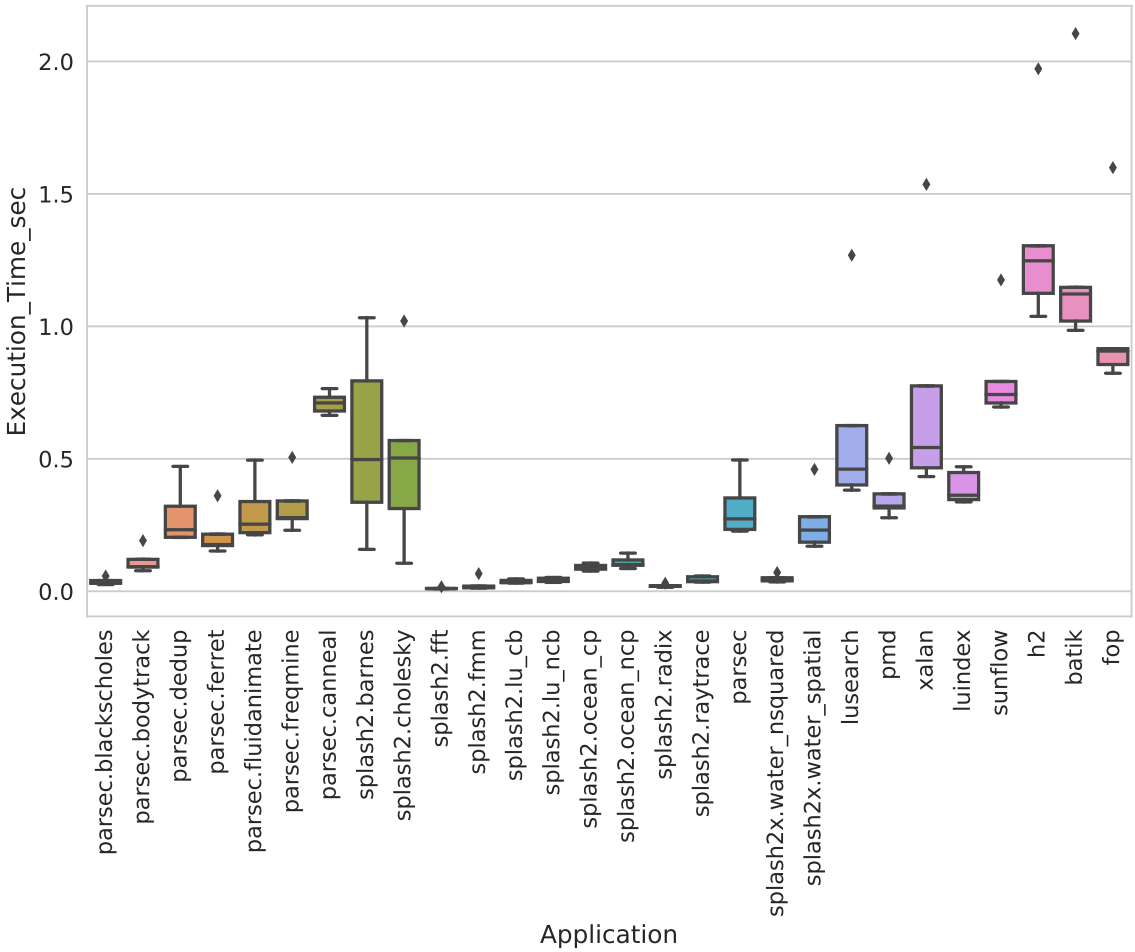

Figure 23: Impact of different resource configurations on the applications' execution time. Variability in the execution time is due to scaling up in the configuration of the resources assigned to the application.

III.7 Related Work

We now compare UPSARA with prior related works. Wienke et. al. [151] have developed a domain-specific language (DSL) for performance profiling, however it is restricted to application profiling only and is coupled to the robotics domain. It utilizes code generation facility to create artifacts required for testing application performance using the Java testing framework.

In [144], a DSL is introduced for web application performance profiling under various load configurations. The DSL automates the cloud resource allocation problem based on the desired QoS goals for the web application. The DSL also generates test cases for load testing the application and monitoring both the system and the application metrics. Similarly, in [54], a DSL called DSLBench is presented. DSLBench generates load testing codes for web application testing. It leverages the Microsoft Visual studio and generates codes in C-Sharp language. AutoPerf [25] presents an automated load testing and resource usage profiling for web service applications. It provides a facility for monitoring resource usage for per request calls in a web session. It also provides a load generation facility.

In [24], an OMG Data Distribution Service-centric performance testing suite is designed leveraging a DSML. It uses model-driven generative capabilities to generate test plans for testing the application's end-to-end QoS under various configurations. It also automates the deployment of the test plans on the cloud environment. Expertus [92] provides an automated performance testing of applications on the cloud environment. It also leverages generative aspects to generate test specification and deployment of the applications using the aspect oriented weaving techniques.

Similar in spirit to our work, [77] presents a declarative DSL and accompanying model-driven framework for automated end-to-end performance testing of container based micro-services. Similar to application profiling which is the focus of our work, performance testing also involves complex set-up of performance tests, configuration

and management of loading infrastructure, deployment under different testing scenarios and post analysis of collected performance data.

While the above DSL tools and methodologies address some of the problems in application performance and system analysis, they still have some drawbacks. In some cases the techniques are tied to a specific programming language, application and/or the runtime platform. As such they cannot be used for heterogeneous runtime platforms and use case scenarios, which UPSARA supports. For example, the solution presented in [24] is tailored towards DDS messaging platform. In [151], performance profiling is geared towards robotic applications running on Java platform. Similarly, web application specific tooling is presented in [25, 54, 144].

Another important differentiation when compared to existing tools is that they do not allow configuring system metrics as captured in the UPSARA DSML. Users still have to write manual configuration scripts for various metric collection probes on the desired runtime platform for performance monitoring. UPSARA's generative, constraint checking and deployment facilities allow for automated synthesis of these monitoring metrics configurations and deployment on the runtime platform.

Moreover, existing tools and approaches lack visual modeling elements which UPSARA's modeling environment provides. This makes configuring of performance monitoring applications much more intuitive thereby presenting a user friendly environment for performance monitoring. UPSARA also creates auto-analysis and visualization notebooks using the Jupyter environment for performance analysis.

III.8 Conclusions

The utility of cloud computing and its services can be significantly improved if performance engineers can rapidly and with ease analyze performance anomalies in cloud-based applications and define appropriate solutions to overcome these problems. Unfortunately, users today face daunting challenges in using existing resource

monitoring and application performance modeling frameworks due to the significant variability incurred across these frameworks in terms of APIs, granularity of monitoring, and making sense of the collected data. To that end this chapter presents UPSARA, which is an extensible platform based on model-driven engineering technology that has the potential to significantly reduce the entry to barrier for performance monitoring and modeling of applications deployed across the cloud, fog and edge resource systems. UPSARA provides high-level, intuitive abstractions which enable users to quickly set up performance experimentation using visual drag and drop components. Generative and constraint checking components within UPSARA ensure that the generated low-level scripts, such as metric configurations, required for the performance experimentation are correct by construction. The configuration deployment component of the framework, e.g., using Ansible [3], satisfies the deployment of the generated configuration artifacts on the runtime platform. The framework also provides the user with an ability to automate the visualization of results and analysis, which enables users to gain deeper insights into the application performance behaviors.

As future work we plan to extend the DSML to support additional kinds of performance testing usecases, such as the effect of hardware configuration, NUMA bounded placement schemes, cache contention scenarios on application performance. We also plan to support experiment workflows that define dynamic operational patterns as seen in real world situations and analyze application and system performance. These patterns include application workload variability, system failure injections, and collocation application execution scenarios.

UPSARA is available in open source at `https://github.com/doc-vu/UPSARA`.

CHAPTER IV

TECHNIQUES FOR GENERIC DISTRIBUTED SYSTEMS VALIDATION FRAMEWORK

IV.1 Introduction

Complexities in Distributed Systems and Algorithms

As the world gets more connected owing to many advances in hardware, software and networking technologies, many new services of significant societal relevance (e.g. healthcare, transportation, avionics, weather prediction, search and rescue, education etc.) are likely to emerge. With the advent of low-cost embedded devices, sensors and other ubiquitous computing devices, such services will be inherently networked and distributed. Although these services will be designed to be easy to use, their underlying design and implementations will be substantially complex. In large-scale networked and distributed systems, many complex issues need to be addressed, such as time synchronization, fault management, replication and replica synchronization, consensus among peers, leader-election among nodes, deadlock avoidance etc. There is also a large degree of heterogeneity in the distributed systems in terms of network topology (ring, star, mesh etc.), node types (fixed vs mobile nodes, static vs dynamic nodes, physical vs virtual nodes), communication types (client-server, peer-to-peer, publish-subscribe etc.), network types (Ethernet, WiFi, Satellite). These design considerations and heterogeneity make the algorithms for distributed systems very complex.

Difficulties in Understanding and Deploying Distributed Algorithms

Existing teaching modalities, tools and techniques for understanding the algorithms for distributed systems often rely on traditional approaches such as didactic

lecturing, simple proof sketches on the whiteboard, and basic simulations or toy assignments. It is general practice in universities to teach distributed systems algorithms theoretically and then having students implement them in a programming language like (Java, Python or C++) or simulate them using basic simulation tools [116]. This approach incurs several difficulties for students including (1) programming them in languages they are not experts in, (2) analyzing these algorithms in simulators/emulators they are unfamiliar with, and (3) needing to deal with accidental infrastructure complexities in order to deploy them on real hardware to realistically validate them or propose improvements and extensions to them. Due to a piecemeal approach in learning and implementing these algorithms (i.e. programming/learning algorithms individually), students (1) cannot analyze multiple algorithms at the same time to compare and contrast them, (2) cannot seamlessly switch between simulation, emulation and real deployment on hardware, and (3) hence do not obtain a holistic view of distributed systems and how different algorithms work together in a real world distributed system.

Solution Approach and Organization of Chapter

To address these challenges, we use Software Product Lines (SPLs) [62] in the context of cloud platforms to improve teaching and learning of distributed systems algorithms. The key intuition behind applying SPL principles stems from the observation that these algorithms tend to share several common traits while differing only in some aspects. Consequently, a collection of distributed systems algorithms can be viewed as variants of a product line. The challenge then lies in understanding and capturing the commonality and variability across these algorithms, and developing techniques needed to automate the synthesis of these variants so that the different dimensions of accidental complexities faced by the student can be substantially alleviated.

Our solution is a framework for distributed systems called the *Playground of Algorithms for Distributed Systems (PADS),* that reifies SPL principles by building on the strengths of model driven engineering (MDE) [134] with generative capabilities, feature modeling and teaching/learning tools & technologies. In this context, this work makes two contributions:

1. We present the underlying feature model that captures the commonality and variability across a collection of distributed systems algorithms, and show how this feature model is realized in a domain-specific modeling language (DSML), as well as generative capabilities that maximally automate the synthesis of product variants (i.e., experiments involving one or more distributed algorithms) that can be deployed and tested at large-scale using cloud platforms.

2. We present a prototype implementation of PADS to showcase a distributed systems algorithm illustrating a peer to peer file transfer algorithm based on BitTorrent, which shows the benefits of rapid deployment of the distributed systems algorithm. Using this example, we provide some qualitative evaluation of PADS showcasing the effort saved on the part of a student.

IV.2 Related Work

In this section we compare PADS to related works along three dimensions: tools for network experimentation, current work in the learning and teaching specific software for distributed systems algorithms, and use of model driven engineering in the design of large scale software systems in the context of learning systems.

1. Tools for network experimentation: The authors in [53] have provided an extensive list of experiment management tools for carrying out research in distributed systems. The strengths and weaknesses of various tools are provided. It highlights the need for good experimentation tools to ensure replicability and reproducibility

of an experiment. It also points out the need for further development of these tools due to numerous challenges in fully exploiting the capacity of certain experimental testbeds. Efforts have been made to address the problem in repeatable research environments. For example, the Apt (the Adaptable Profile-driven Testbed) approach has been presented in [132]. It builds an experiment profile which describes the dependencies needed to conduct networking experiments. Dependencies include both hardware and software requirements of the experiment. Researchers can build their own testbeds and share these profiles with others, who can then repeat and reproduce the experimental environment.

The cOntrol and Management Framework (OMF) [128] provides a modular architecture for managing heterogeneous resources in networking testbeds. It manages resources such as remote bootstrapping, and saving and loading disk snapshots for conducting experiments. It also features an experiment description language which can be used to specify resource requirements and configuration and experiment orchestration. OMF-F [127] is based on OMF [128]. It addresses the shortcomings of OMF which was targeted for a single testbed deployment only. OMF-F allows management of resources for network experiments across federated networking testbeds. It provides a DSML supporting special event-based experimental scenario. It also supports management of resources using a resource model which features publish-subscribe messaging pattern for communication and control of resources. Further, it supports scalable deployment of experiments across federated testbeds.

The above frameworks are oriented more towards the research community to conduct experimental research. Unlike these tools, PADS is designed as a learning aid in teaching distributed systems algorithms. Our approach also facilitates use of both the simulator and real world testbed in a single development environment for testing different distributed systems algorithms. We also use the SPL approach in the design and development of PADS.

2. Learning systems for distributed algorithms teaching: Authors in [66] present a comprehensive survey providing an overview of different tools, simulators and learning platforms available for teaching distributed systems. It outlines tools available for managing deployment, execution, discovery, monitoring and configuration of distributed systems. It also presents a list of algorithms that can be used for teaching and demonstrating intricate details of distributed algorithms.

ViSiDiA [21] is a framework for designing, simulating and visualizing distributed algorithms. It is developed using JAVA frameworks. It provides implementations of different distributed systems like sensor networks and mobile agents. A user can specify their custom distributed algorithms by making use of framework specific JAVA API. Distal [45] is another framework that is specifically aimed at a certain class within the distributed systems algorithm, namely fault-tolerant systems. It is developed on top of the Scala programming framework. One can write pseudo code for the algorithm using its DSML to translate into an executable code. The executable can then be deployed on clusters for testing. It lacks integration with simulators that would facilitate quick testing and debugging of algorithms.

LYDIAN [98] is an animation environment for visualizing the behavior of distributed algorithms. It supports writing custom protocols which can be executed on the LYDIAN's simulator and the output animation can be viewed on the TCL/TK based graphical user interface. It also supports playback of algorithm execution events using a trace file for quick demonstration of algorithm behavior on the animation windows. VADE [114] is another framework that provides visualization of distributed algorithms. The VADE framework is designed such that computation and implementation of algorithm is done on the server side while users can see the algorithm visualization via JAVA applets using the web browser on their client machines. The algorithm is implemented using JAVA programming language.

Another teaching and learning framework called FADA (Framework Animations of

Distributed Algorithms) is presented in [120]. In FADA, the simulations are written using JAVA programming language using the visualization APIs provided by the framework. It also provides a set of preassembled simulations for different algorithms which can be used as examples for demonstrating distributed algorithms to students.

The frameworks presented above have the following shortcomings compared to our approach. First, the distributed algorithms need to be written in a language which the framework supports. Secondly, the tools presented above do not support seamless translation of programming artifacts from simulation to real world deployment.

3. MDE in learning systems: Previous work has shown MDE and DSML being effective tools [115] in developing teaching software systems. Students have also seen the benefits of rapid code generation based on MDE techniques. In [80], students were able to rapidly synthesize code artifacts using MDE to rapidly generate code, when changes where required to be made in platform configuration of robotics control code and mobile device.

An educational game design software framework is presented in [95]. It utilizes a model driven approach to describe educational game concepts. It presents an educational game metamodel that defines platform-independent educational game concepts. The framework aims to design educational games to motivate the students to get involved in the learning process thereby effectively conveying educational material.

SPL techniques were applied for design, development and support of a family of elearning systems [59] called TALES. It also highlighted some of the challenges involved in the development of large-scale educational system and how SPL helped it to gain 10-fold productivity boost in the developmental efforts. The educational systems were built as a part of Adult Literacy Programme (ALP) for teaching illiterates in India in 22 Indian languages. Unlike the work presented above our area of study is focused on a special topic within computer science which is the distributed systems

algorithms. Our work leverages the MDE and the SPL techniques in the design of a learning framework for distributed algorithms.

IV.3 Dimensions of Variability in the Development and Deployment of Distributed Systems

Decentralization is a fundamental aspect of distributed algorithms. Distributed systems algorithms deal with a large number of issues that arise in distributed systems, e.g., challenges pertaining to successful coordination of various entities to address system fault-tolerance, communication heterogeneity, and distributed time synchronization. Given these challenges, distributed algorithms are difficult to comprehend and its implementations are often non-trivial. Moreover, these algorithms target different classes of problems such as consensus, synchronization, discovery, fault-tolerance, performance and correctness. To observe the behavior of a distributed algorithm in action one could use a simulation environment or real time observation on a set of actual distributed systems. Network simulators can be used to implement and test existing or new algorithms in a controlled environment at a lower expense both in terms of time and money.

This section elicits the key challenges in the learning process of distributed systems. It presents a motivating scenario based upon which a set of challenges are documented that subsequently drive our research on PADS.

IV.3.1 Motivating Scenario

Figure 24 shows a motivating scenario illustrating a workflow that captures a typical approach to hands-on learning of distributed systems algorithms. A number of challenges manifested in this workflow and described below prompted us to investigate solutions to address these challenges.

As shown in Figure 24, teaching and understanding of an algorithm is typically

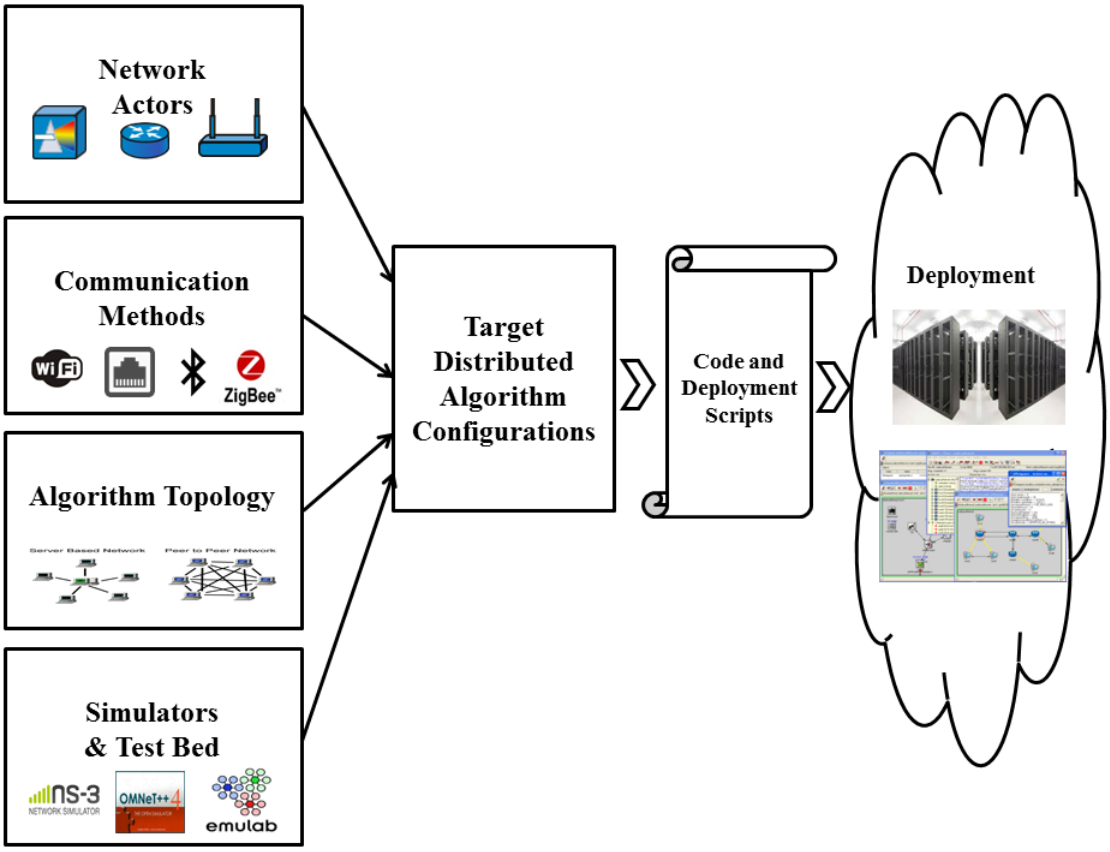

Figure 24: Commonality and Variability in Algorithm Design and Deployment Workflow

accomplished by simulating the algorithm in simulators that can be parametrized in a variety of ways or by running the algorithm in testbed environments that can be configured in multiple different ways. This apparently simple teaching and learning workflow manifests a significant amount of variability both within and across the stages of the workflow. For example, many different network simulators exist such as $OMNET++$ and ns-3. Similarly many different network testbeds exist such as Emulab, PlanetLab and GENI. Furthermore, the algorithm demonstration involves setting up of different deployment configurations and network topology setups. The algorithm under study may involve a variety of network actors like mobile hosts, clients, servers, peers, elected leader actor, and network coordinator. The underlying network communication may involve different types of network interfaces either

standalone or a combination of Bluetooth, WiFi, ZigBee, Ethernet, USB, and serial interfaces. Moreover, the target algorithm involves defining the network topology for demonstrating the experiment and understanding the algorithm details.

As an example, the process of understanding a peer to peer file transfer algorithm such as BitTorrent will involve a set of peers which communicate with each other. In this scenario, there would be different network actors like Peers and Trackers. These actors, which are specified by the algorithm, may communicate via an Ethernet interface in a LAN environment or over a wireless link. The network interface could support variable transmission speeds. For demonstrating the target algorithm, one also needs to represent the desired network topology accounting for all the associated network actors in the system thereby ensuring a faithful operation of the algorithm. Based on all the configurations chosen in the different stages of the workflow, a student must then proceed to deploy the experiment in the deployment environment, which can be either a simulator or a testbed.

IV.3.2 Challenges and Requirements

As evident from the above scenario, any tool used in learning and deployment of distributed systems algorithms must manage a large amount of variability across the different stages as well as support the key commonalities. The commonalities include the communication model, model of computation, and the problem being solved, e.g., consensus, fault tolerance, consistency, lookup, etc. We surmise that by capturing the variability in the demonstration of the distributed algorithm as a software product line would enable instructors to rapidly provision a desired target algorithm, which in turn will make it easier for users to learn it thereby saving time in the experiment configuration and setup. Based on the challenges discussed, we summarize following requirements for PADS.

Requirement 1→ Extensibility and Tool Reuse: The tool should enable

new algorithms to be added to the existing collection. At the same time, any associated tool such as a simulator or a testbed must be reusable in the context of newly introduced algorithms. At the same time as new simulators, experimental testbeds, and real-world scenarios emerge, the framework should be extensible to include these new back ends also.

Requirement 2→ Programming and Deployment heterogeneity: The framework may be tied to one programming language but the student may not always be familiar with that programming language. One should be able to program the distributed systems algorithms in the language one is familiar with and the SPL framework should support learning of distributed algorithms by being implementation language-agnostic. The deployment of algorithms can be made on simulators, emulated testbeds or in a real world network deployment. The SPL framework must allow users to define a network topology of the target distributed algorithm and run it on the desired deployment environment thus enabling the possibility to seamlessly transition the code artifacts written for a simulator to a real-world deployment or vice versa. There should be clear separation of concerns between the definition and deployment of network topology.

Requirement 3→ Integrated tool to rapidly create topology and deploy experiment: It is a challenging task to use existing tools to rapidly create a new network topology to showcase the desired distributed systems algorithm and deploy on single or multiple deployment environments all using a single toolchain. It is hard to find such tools that can be used to run experiments targeting multiple deployment frameworks in an integrated tool suite. Such a requirement must be met.

IV.4 Design and Implementation of PADS

We now discuss the design and implementation of the Playground of Algorithms for Distributed Systems (PADS), which is an extensible framework that manages a

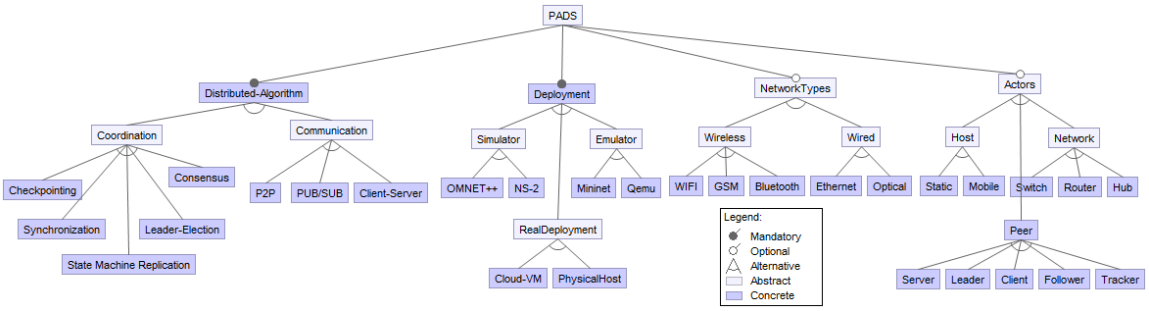

Figure 25: Feature Model Diagram of Playground of Algorithms for Distributed Systems (PADS) Framework

software product line of distributed algorithms used as an instructional and learning aid for distributed systems. It uses MDE and SPL techniques to integrate various distributed systems algorithms for teaching, and cloud platforms for deployment of experiments. We also show how PADS addresses the challenges and requirements introduced in Section IV.3.

IV.4.1 Feature Model Representation

For a successful SPL for PADS we need to manage the commonalities and variabilities that are exhibited for realizing the development, implementation and demonstration of distributed algorithms. One of the well known approaches for representing and managing these commonalities and variabilities is by the means of feature models [40]. Feature models provide proven techniques for improving reusability by specifying the reuse rules.

The feature model for PADS must capture the commonalities and different dimensions of variabilities that we highlighted in Section IV.3. To that end we have defined a conceptual feature model for PADS shown in Figure 25. The `PADS` framework is represented as the root feature in the figure. The `Deployment` and the `Distributed-Algorithms` are the required features. Both these features are required

because we must have an algorithm that we want to test, and it must have one of the many potential choices available for deployment so that it can be tested.

These features are used to capture from the user the types of target distributed systems algorithm to be evaluated and where to perform the deployment of such algorithm. The `Deployment` can be made on `Simulator`, `RealDeployment` or an `Emulator`. `OMNET++` and `NS-2` are types of `Simulator` feature. `Emulator` can be `Mininet` or `Qemu` which are type of emulator tools available for experimentation. `RealDeployment` can be done on cloud based virtual machines(`Cloud-VM`) or on actual physical host machine(`PhysicalHost`) to conduct experiments. The `Distributed-Algorithms` consists of an extensible set of distributed algorithms, which in turn can be categorized under different classes, such as `Coordination` and `Communication`. `Coordination` category of algorithms include `Checkpointing`, `Synchronization`, `State Machine Replication`, `Consensus` and `Leader-Election`. Peer to peer (`P2P`), publish-subscribe (`PUB/SUB`) and client-server (`Client-Server`) communication models are a part of `Communication` models category. The `Distributed-Algorithms` comprises one or more kinds of `Actors`. `Actors` may be an abstract representation of peer systems: like `Server`, `Client`, `Leader`, `Tracker`.

Other actors could represent an abstract notion of the host system, which could either be `Mobile` or `Static` host. Some of these actors could represent network devices based on the functionality it performs like `Router`, `Hub`, `Switch`. Each of these `Actor` feature exhibits a set of network and communications features. These features are represented by the `NetworkTypes` feature. `NetworkTypes` could be `Wireless` or `Wired` communication interface. `WIFI`, `GSM`, `Bluetooth` are a type of `Wireless` communication interface. `Ethernet` and `Optical` are type of `Wired` communication interfaces.

IV.4.2 Realizing The PADS Feature Model Using Model-driven Engineering

The SPLs can be built using modular software. Changes in the feature configurations can be mapped to the changes in the software modules [150]. Design and development of modular software framework is a challenging task. We use model-driven engineering(MDE) techniques to codify the feature model by mapping it to metamodel(s) of a domain-specific modeling language and use generative technologies, which are key artifacts of MDE, to automate the synthesis of product variants of our PADS product line.

Our PADS framework is hosted on virtual machines deployed on Openstack cloud, which is an open source cloud computing infrastructure [146]. Cloud computing provides on-demand access to large pool of shared resources for compute intensive simulations and network experimentation.Users can access these virtual machines using remote access client which could be either a web browser client or a desktop client.

Figure 26 shows the overall process of creating a network topology, selection of the distributed algorithm, intermediate topology-specific code generation, and deployment of the algorithm using our PADS framework. A user develops a model of the experiment using our DSML. The user first selects the algorithm. The algorithm selection can be done from one of the pre-assembled algorithm modules provided as part of the framework. The user then creates the network topology for the desired test algorithm. The framework has built-in constraint checker module which verifies if the test topology is supported, if any violations are present it will notify the user about the constraint invalidation. Next, to conduct the experiment the user selects the deployment environment. Once the experimental setup for a given algorithm and deployment environment are modeled, a set of reusable model interpreters are

executed to automatically generate the deployment engine specific glue-code and the execution artifacts.

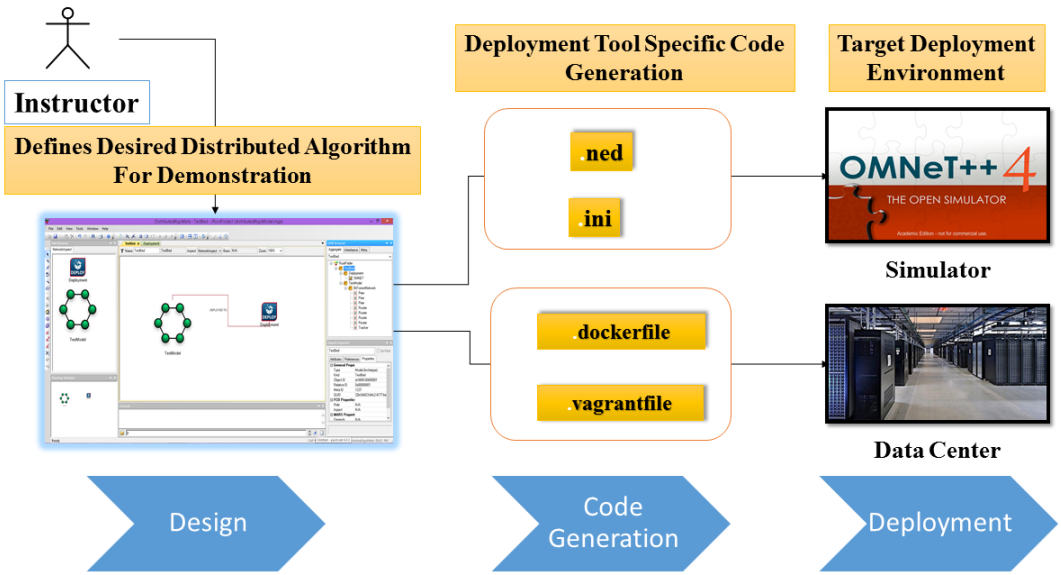

Figure 26: Model-based Process for Distributed Algorithm Demonstration and Deployment

The Generic Modeling Environment (GME) [101] is used to develop the DSML and generative capabilities for provisioning and deployment management of the experiment. GME provides an environment to define the syntax and semantics of a DSML through metamodeling. Model interpreters can be defined associated with the metamodels that can provide additional semantics to the language which are not captured in a visual form as well as provide the generative capabilities needed for automation. The same GME environment can be used to build model instances of a DSML. Thus, in our PADS framework, an users can extend existing metamodels for the collection of algorithms by providing a metamodel for a new algorithm. Next, the user can then use the PADS framework to develop model instances and configure them for their experimental scenarios.

Figures 27, 28 and 29 illustrate the metamodels used in the framework. In these metamodels we have mapped the features that we described in the feature model into concrete metamodeling artifacts. The metamodel primarily comprises first class entities, such as DistributedAlgorithm, Deployment, and Main.

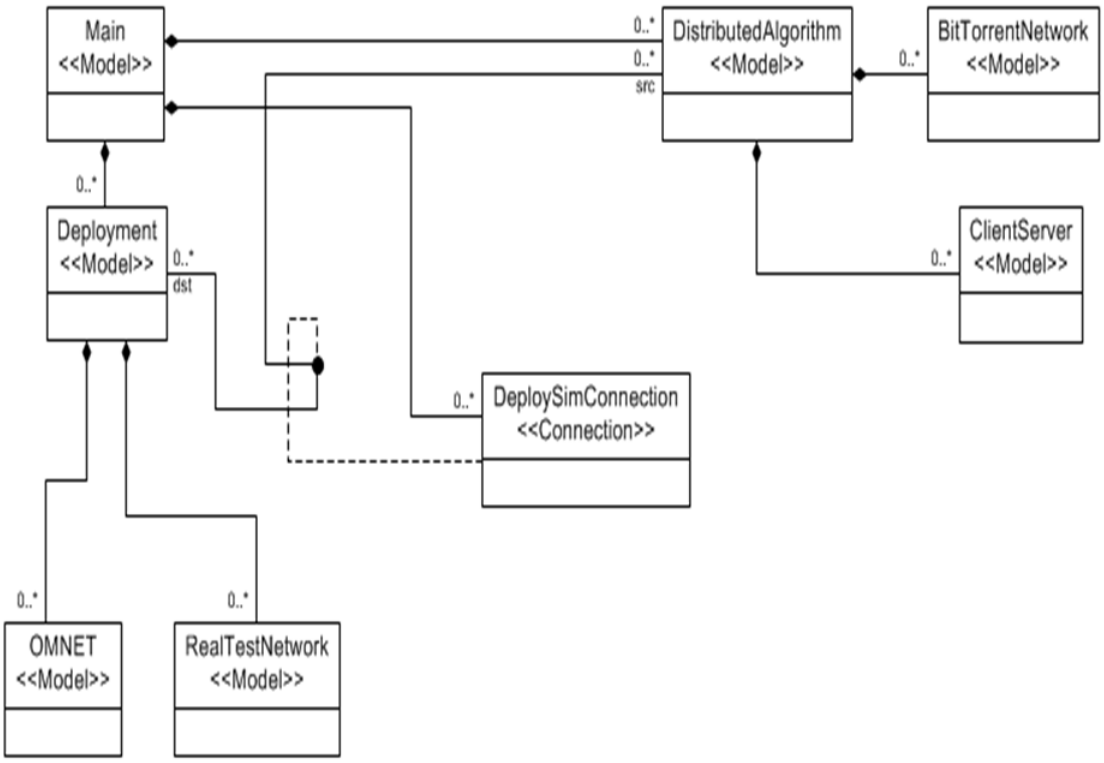

Figure 27: Meta-Model of Playground of Algorithms for Distributed Systems (PADS) Framework

Next, we describe the different metamodel components in the DSML and their responsibilities:

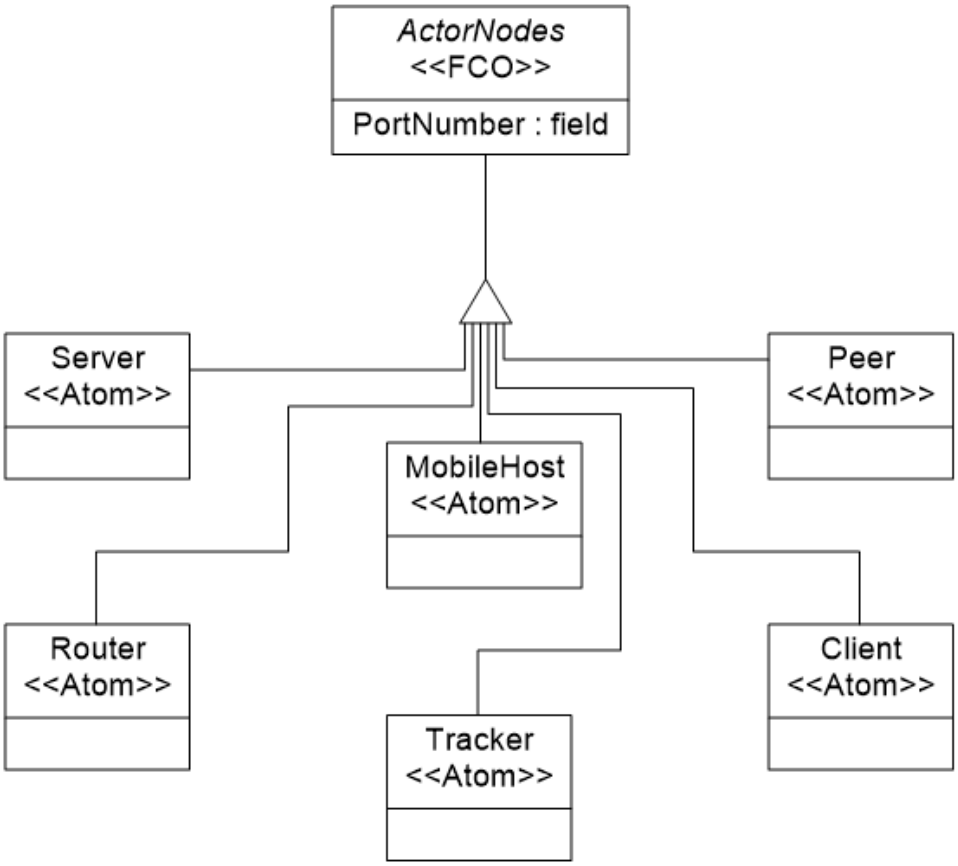

Figure 28: Meta-Model of Network Actors

- *Main:* represents the main meta-model block of PADS framework. This component aims to provide information for all the framework components like: *Deployment* and *DistributedAlgorithm*, which can be configured by users depending on their experimental setup needs.

- *DistributedAlgorithm:* defines the distributed algorithms supported by the framework. Users can then select distributed algorithms contained within this model. As can be seen in the Figure 27, the *BitTorrentNetwork* and *ClientServer* algorithms are currently available with *DistributedAlgorithm*.

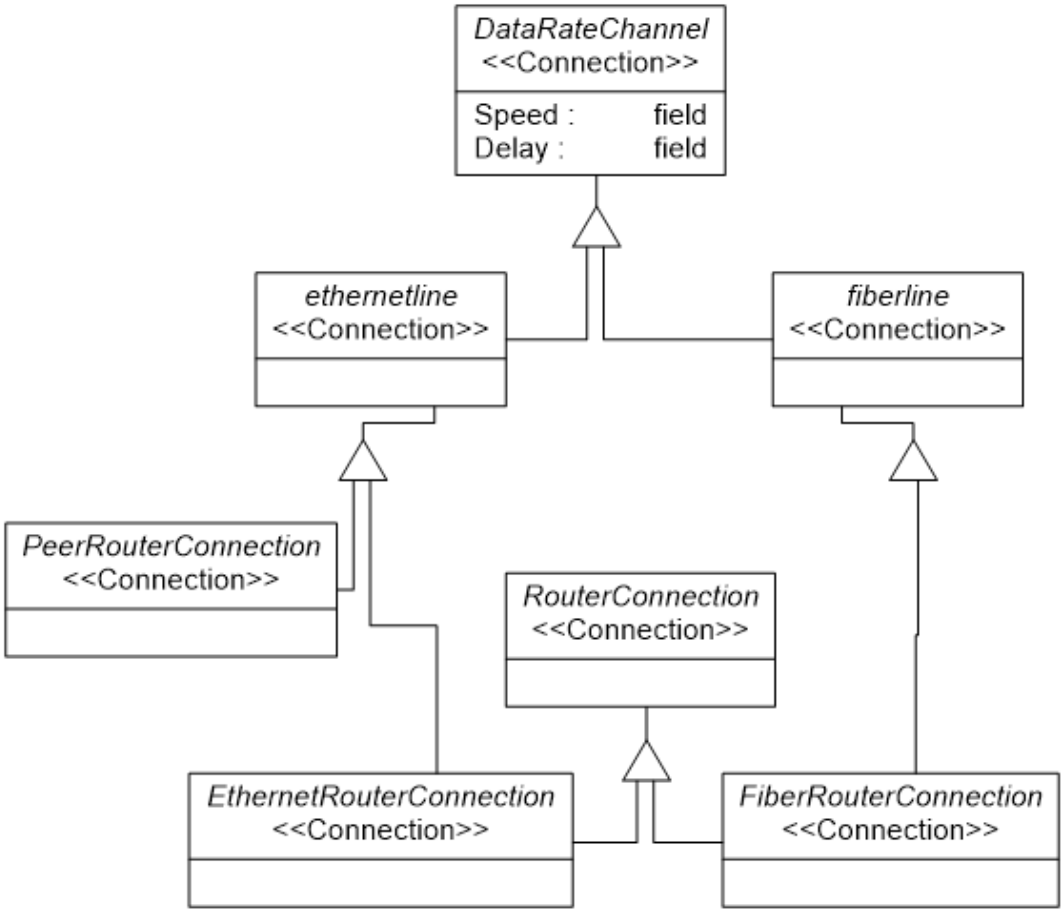

Figure 29: Meta-Model of DataRateChannel representing the communication characteristics

- *Deployment:* specifies the type of deployment environments available for testing the algorithms from *DistributedAlgorithm*. A user can create the network topology and deploy the algorithm on the target environments supported by *Deployment*. *OMNET*, which is a simulation environment, and *RealTestNetwork*, which is a real world testbed, are two such deployment environments.

- *ActorNodes:* these represent the type of network participants/nodes that will be used in the algorithm. As seen in Figure 28, these have an attribute field of *PortNumber* which represents the network port of the node. *Server*, *Router*,

Peer, *MobileHost*, *Tracker*, *Router* are such *ActorNodes* that can be utilized in distributed systems algorithm as network actors.

- *DataRateChannel:* is used to define the network communication characteristics. Communication properties like communication rate (*Speed*) and communication delay (*delay*) can be specified for the network. *Ethernetline* and *Fiberline* are two such communication media represented in the Figure 29.

- *DeploySimConnection:* acts as a bridge between the test algorithm and the deployment environment. Therefore the *DistributedAlgorithm* defined by the user are connected to the *Deployment* component via *DeploySimConnection.*

Our framework also leverages the GME's OCL constraint checker facility to detect design time configuration errors. If there is any violation, then the constraint checker reports this violation to the user.

In Figure 30, an example model of execution of a distributed systems algorithm using the DSML is illustrated. In the figure, DSML components such as `TestModel` (*DistributedAlgorithm*) and `Deployment` (*Deployment*) are defined. In this example model, we have selected *OMNET++* as a target deployment environment. Selection of desired distributed algorithms is done as shown in the model in Figure 31. Here we have selected a BitTorrent algorithm as an example.

IV.4.3 Meeting the Requirements

1. **Extensibility and Tool Reuse:** The framework supports working with new distributed systems algorithms and topologies. Using the MDE concept of meta-models, we can specify the metamodels for a new distributed systems scheme. This also applies to new deployment environments which the framework may need to support in future. As shown in Figure 27, we can add new distributed systems algorithms and deployment schemes by attaching the new meta-models

Figure 30: Model example

to *DistributedAlgorithm* and *Deployment* metamodels, respectively. We also need to specify the model interpreter and how to generate the new configuration code for the new metamodels. Thus this framework can be easily extended to support new type of distributed systems and deployment environments on top of the existing software tools.

2. **Overcoming Programming and Deployment heterogeneity:** We leverage MDE technique where in we created a DSML which allows one to describe the type of distributed systems one would like to experiment with. Using the DSML the user can create a network topology of the distributed systems and deploy it on the target environment to which he/she is familiar with. For example, if the user is familiar with the programming construct of OMNET++ simulator, the framework will autogenerate configuration files which are compatible with

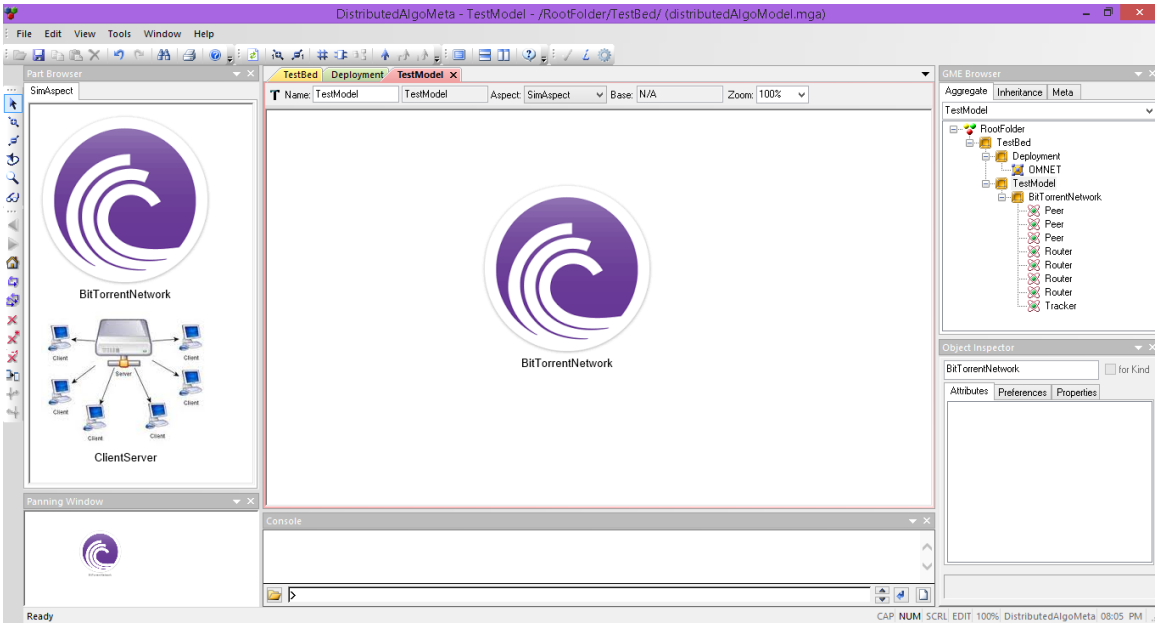

Figure 31: Model of target algorithm selection

OMNET++ environment. If an experiment needs to be conducted on the cloud, the tool will generate the files required to configure and deploy the experiments on virtual machines in the cloud.

3. **Integrated tool to rapidly create topology and deploy experiment:** This framework is designed to facilitate users to use a single toolchain that can create and deploy large distributed systems on different deployment environments. It also lets a user reuse the same network system topology and allows to target simultaneously to multiple deployment environments. Also the auto-generation code facility helps one to quickly configure new network topology without manually writing the network configuration code which can be both tedious and error prone.

IV.5 Framework Validation

In this section we evaluate PADS along three dimensions. First, we show how PADS can be extended to include a new algorithm. Second, we show PADS' effectiveness in terms of effort saved on the part of the user in using the framework. Third, based on the case study used in the first two evaluations, we elicit how the three key requirements from Section IV.3 are met by PADS.

IV.5.1 Extensibility of PADS

To showcase the extensibility of our framework we have created an application based on a peer to peer (P2P) distributed system that uses the BitTorrent algorithm [124]. The BitTorrent system is a P2P distributed system that facilitates downloading of files. This is achieved by splitting a file into large number of chunks, which may be spread across a number of peers. A peer interested in downloading a specific file can download the file by downloading different chunks simultaneously from peers who have those chunks. The BitTorrent algorithm has an elaborate scheme involving a Tracker node which keeps track of peers in the system and what chunks they hold.

Thus, in the BitTorrent system, there are two kinds of network actors: a *tracker* and *peer*. As mentioned above, the *Tracker* is a centralized entity responsible for keeping track of the location of file copies in the P2P network. It also keeps track of available file chunks with the participatory peers in the network. Typically *peers* that are interested in downloading a certain file query the *tracker* for the availability of the file. Once the peer has the information required to download the file chunks from the tracker, it downloads the file chunks directly from the respective peers without the tracker acting as a broker. Moreover, peers that are interested in sharing the file register themselves with the tracker so that other peers interested in downloading that file can find them.

To enable the BitTorrent systems logic in our framework the user can first specify a meta-model as shown in Figure 32. It consists of a *peer*, *router* and *tracker* network actors in this *BitTorrentNetwork* model. The proxy elements in the figure basically refer to the references to their target models. We also see different kinds of *DataRateChannel* connections depending on the type of link shared among the network actors.

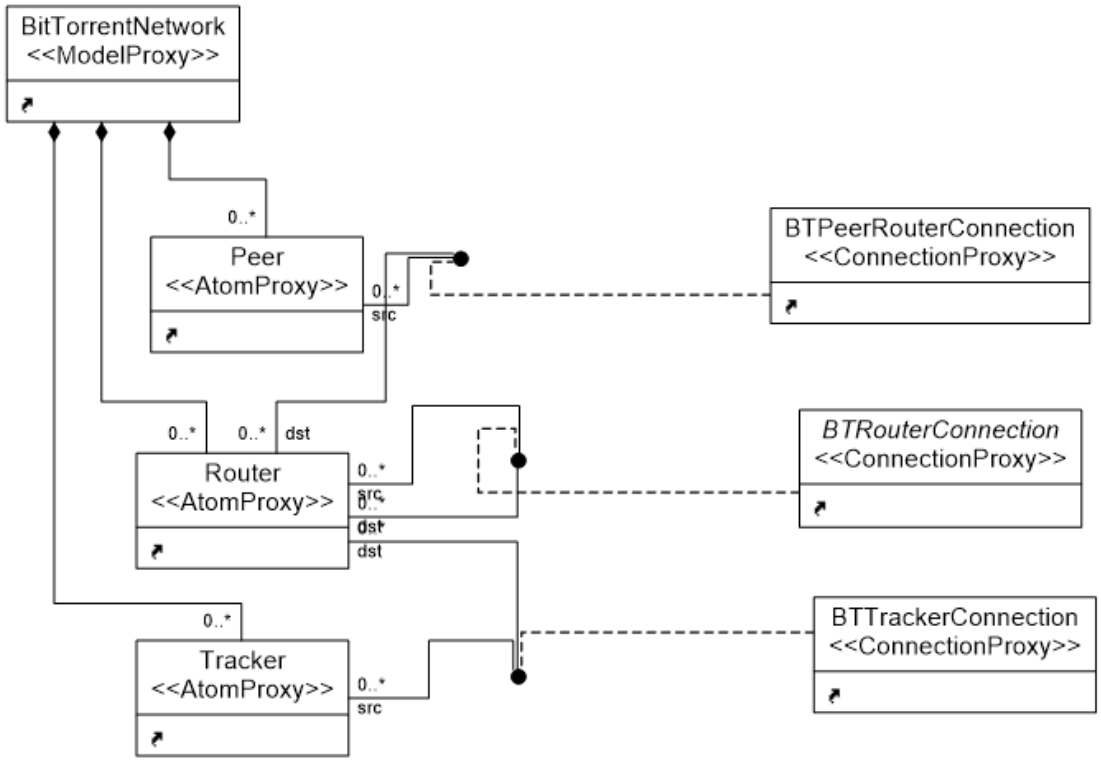

Figure 32: Meta-Model of BitTorrent Algorithm

Figure 33 provides more details about how the specific connections such as *BTPeerRouterConnection*, *BTRouterConnection* and *BTTrackerConnection* are derived from the base *DataRateChannel* model. OCL constraints for the metamodel which ensures that there is at least one *peer* and only one *tracker* in the deployment experiment as shown in the Listing IV.1 can be specified by the user. Next, the model instances can be created to create complex topologies and can then be deployed.

Figure 33: Metamodel of BitTorrent Network Connection

Listing IV.1: Using OCL to Detect Conflicts in BitTorrent System Modelling

```
-- onError: "Network Topology needs at least one Peer and
   Tracker"
self.atoms("Peer") -> size>1
self.atoms("Tracker") -> size=1
```

- Using the graphical modeling feature the user can choose to first draw the connection between *DistributedAlgorithm* and *Deployment* as shown earlier in Figure 30.
- From the *DistributedAlgorithm* the user can select the *BitTorrentNetwork* model as shown in Figure 31.
- Next the user can configure the experimental topology using the *Tracker*, *Peer*

and *Router* from the selection window. One such possible topology is shown in Figure 34.

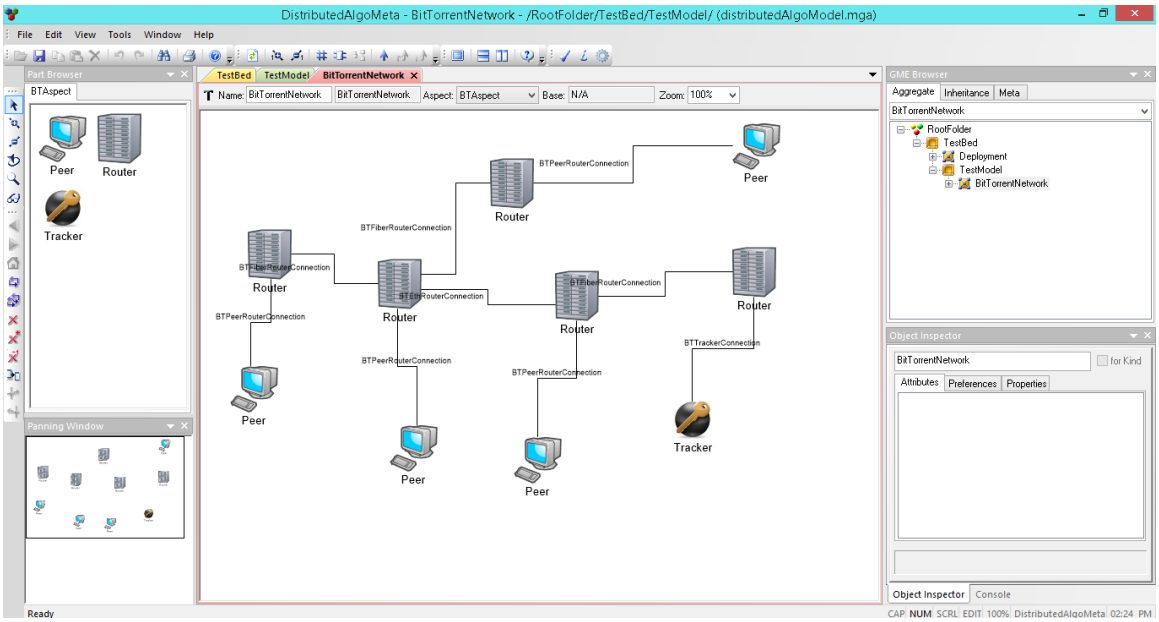

Figure 34: Model of BitTorrent Algorithm

- The user must also specify where he/she would like to deploy the experiment. Using the *Deployment* model the user selects the target deployment.

- The user can run the OCL constraint checker to validate if there are any constraint violation in their model selection.

- Once the experimental model is specified, the GME interpreter is invoked which generates the intermediate code for the network topology which is specific to the deployment environment. Figure35 shows the screenshot of the source code section auto generated.

- Further, the GME interpreter calls the deployment runtime engine and passes

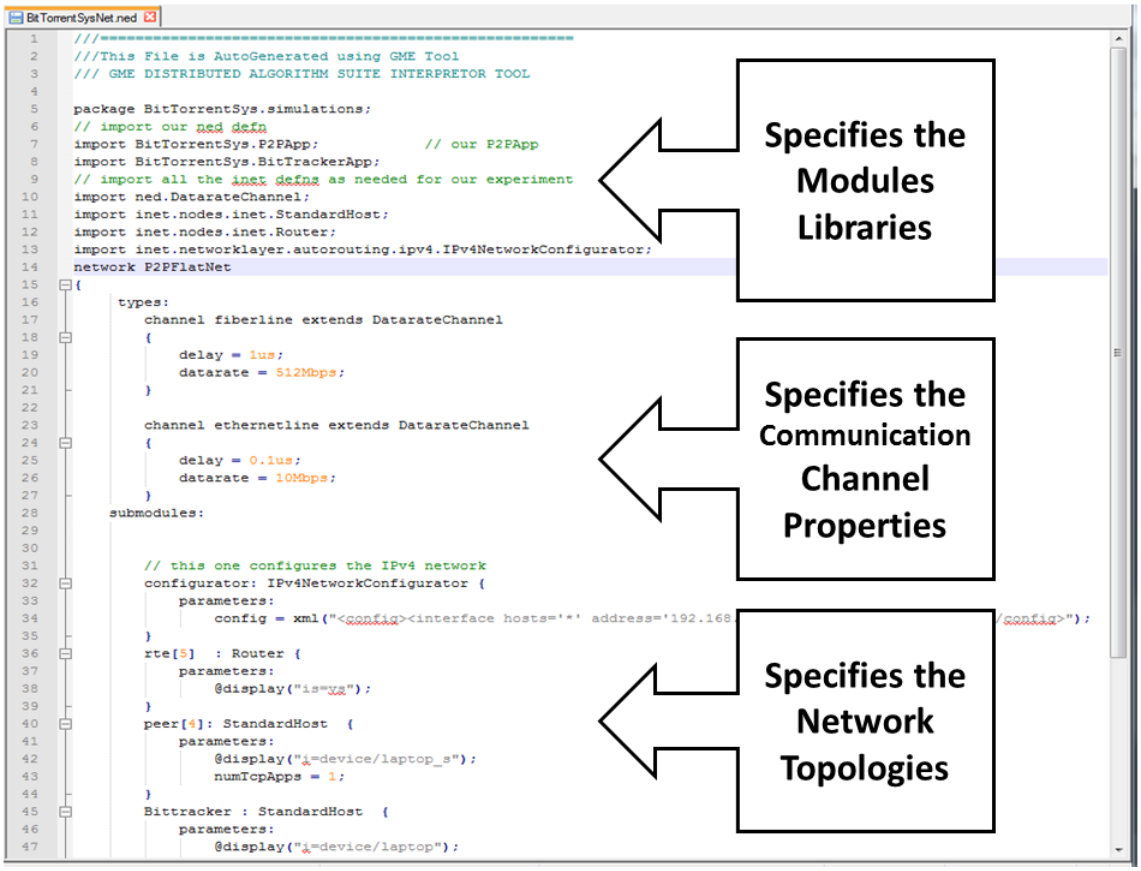

Figure 35: Screenshot of the section of code generated by GME interpreter

the generated code and executes the experimental topology in the target environment. The user can then analyze the results and performance of the distributed systems using the tools provided by the deployment environment.

IV.5.2 Effectiveness of PADS

In this section we describe our evaluation of Distributed Systems Playground. We focus on how the framework alleviates the error prone and tedious effort of manually writing the network topology and deployment configuration code. We then summarize how our approach addresses the challenges discussed in the section IV.3

To evaluate how the modeling approach helps in simplifying manual configuration

glue code of the network actors used in our Distributed Systems Playground, a topology of a distributed systems similar to Figure 34 is constructed. It consists of varying number of routers and peers. For each such configuration we record the number of lines autogenerated by the GME interpreter. As can be seen from the Table 3, the number of generated lines increases with an increase in the number of network actors. One can observe the effectiveness of such autogeneration feature specifically in the context of large distributed systems topology, without which one would need to configure all the connection links and endpoints manually. Manual configuration of such large system can be very tedious and error prone. The Distributed Systems Playground automatically generates all the configuration glue code which is deployment environment specific, thereby simplifying modeling significantly

Table 3: Increase in the number of generated code lines with increase in number of components

Router	Peers	Total Lines
5	4	85
10	103	190
20	203	299

IV.5.3 Meeting the Requirements

Based on our BitTorrent case study from above, we now qualitatively describe how PADS meets the challenges and requirements discussed in Section IV.3.

1. **Extensibility and Tool Reuse:** Section IV.5.1 demonstrated how user can use the existing PADS framework and introduce a new algorithm. It also explains how a user can then model the system and use all existing back ends provided by PADS.

2. **Overcoming Programming and Deployment heterogeneity:** Sections IV.5.1

and IV.5.2 show how a user can model an algorithm and with a click of a button, most artifacts needed to experiment with the algorithm are generated thereby relieving the user from having to deal with any programming and deployment heterogeneity.

3. **Integrated tool to rapidly create topology and deploy experiment:** Sections IV.5.1 and IV.5.2 also illustrate how easy it is to extend the framework and rapidly create the experimental scenarios and test them in a variety of deployment scenarios.

IV.6 Concluding Remarks

This chapter motivated the need for an integrated framework used for demonstrating distributed systems algorithms. The genesis of this work stemmed from our experience as an instructor and student of a Distributed Systems course where a number of different algorithms were studied. We were interested in developing a learning aid to overcome the challenges we faced in the course. Our software engineering background helped us develop an intuition behind our solution approach. We realized that the problem space can be viewed as a software product line because of the commonalities and variabilities demonstrated by the problem space and how individual algorithms and their testing environment can be viewed as individual variants or members of a product line.

To realize these ideas, we decided to utilize a model-driven engineering approach comprising metamodeling and generative techniques. Modeling based on visual artifacts provides intuitive means to model a system using artifacts that are closer to the domain in which the system is being modeled while generative mechanisms automate many of the mundane and repetitive manual tasks. To that end, the initial prototyping of the ideas were accomplished as a class project for an MDE course. Our cloud

hosted solution called the *Playground of Algorithms for Distributed Systems (PADS)* is a continuation of these original efforts.

PADS provides intuitive, domain-specific modeling abstractions to capture various distributed systems algorithms' components and requirements. The playground resolves the potential conflicts faced by learners/researchers, such as programming these algorithms in simulators they may not be familiar with, or implementing them in specific programming languages and deploying them on real testbeds. The playground provides fundamental distributed systems building blocks that a student can use to model their system, and automate the tasks of generating simulation or real-world code, deploy these on the platforms, visualize the resulting behavior and provide feedback to the user so they can continue to iterate through this learning cycle. The playground has an extensible interface and as such has lot of capabilities for adding and supporting both various distributed algorithms and deployment plans.

We evaluated the capabilities of PADS using a representative case study of BitTorrent, which is a peer to peer file sharing distributed system. Our evaluation indicates that it prevents designers from making errors in the distributed systems algorithms test-bed setup and significantly simplifies system deployment by automating the generation of platform-specific metadata that faithfully implements the necessary execution dependency.

Our future work will involve the following dimensions of work:

- **Extensibility:** In its current form, the suite of algorithms we currently have in our PADS framework is rather limited (e.g., BitTorrent, Chord, Paxos). We aim to improve the collection by adding several new algorithms. Moreover, currently our communication model is restricted to TCP/IP level communication. Many higher level protocols and systems hide these low-level details and instead offer other means to represent the end points. We plan to identify this variability in the problem space.

- **Reuse:** We do not aim to reinvent the wheel. Many other frameworks exist that tend to provide specialized capabilities. For example, frameworks such as Ptolemy provide effective mechanisms to model different models of computation. We will seek solutions to integrate such frameworks within PADS.

- **User studies:** Utilizing the framework in an actual course and evaluating its effectiveness in impacting the learning outcomes is a significantly important activity. We plan to undertake this activity in future offerings of the course. By opening up the framework for downloads, we also hope that others would utilize its capabilities and report on its effectiveness.

CHAPTER V

DESIGN STUDIO FOR CO-SIMULATIONS

V.1 Introduction

With a rapid growth in mixed energy generation technologies such as wind and solar energy, the generation and distribution of energy is moving from a centralized grid to a more distributed paradigm. Traditionally, in the energy grid, the flow of energy distribution was from the main power grid operator to the consumers. However, due to increasing affordability of renewable energy harvesting equipment such as solar panels, there is an influx of energy produced by traditional utility consumers, who are feeding the excess energy back to the power grid. The presence of these new actors, also known as prosumers, will only accelerate as the cost of generating energy from these distributed energy resources (DER), such as solar, wind, and biomass, further decreases with advancement in efficient solar panels, energy storage technologies, and related power systems techniques.

With the rise in these mixed sources of energy, the future smart grid systems need to manage distribution of the power flow around the system. The distribution system operators (DSO), thus will play a crucial role and will need to intelligently manage the power demand and supply balance. To provide the best service, a DSO needs to aggregate real-time information about the local power demand and as such needs resilient communication infrastructure to read from all the smart energy meters and to have control on the network flows. DSOs can also set billing rates depending on the power demands. Market analysis thereby becomes more significant to the DSO who will want to set dynamic pricing for the power, in order to accommodate fluctuation in energy demands on a daily and seasonal basis. Since DSOs also have the authority to set prices to buy and sell power from the local prosumers, the role of

market regulator also becomes crucial, so as to not have any DSO monopoly over the power market. Market regulators are thus responsible for maintaining transparency and competitiveness in the local energy market.

Thus, in the mixed energy smart grid system there are many stakeholders, who are directly responsible in successful operation of the smart grid system. Figure 36 illustrates some of these stakeholders in the system. As can be seen, for a successful smart grid system, domain experts representing various stakeholders need to holistically analyze and study the system.

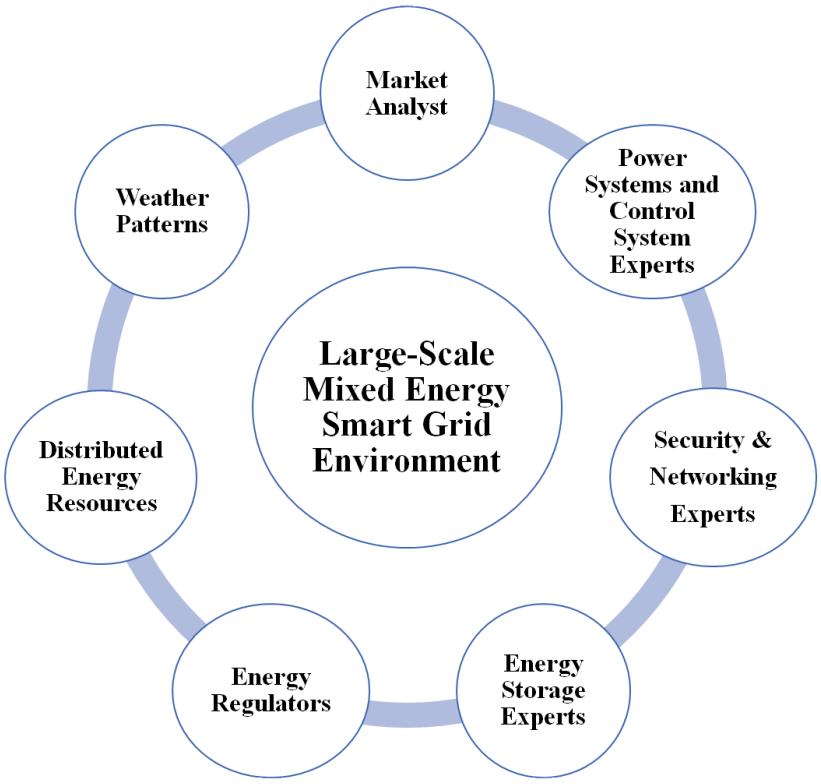

Figure 36: Stakeholders in a Mixed Energy Smart Grid System

An approach to studying such large-scale complex systems is by means of simulation. However, smart grid systems are composed of multi-domain subsystems comprising security, DER, cyber-communications, control systems and electric grid.

Thus, one needs a simulation and analysis tooling infrastructure that spans these multiple domains for designing and simulating such large-scale systems. Despite advances in the simulation tools, a single simulation tool is not able to capture and simulate these various physical and other system aspects of these multi-domain systems. Thus *co-simulation* or coupled simulation have gained popularity in bringing together simulation tools of different domains to simulate scenarios as in the case of mixed energy smart grid system.

As there are multiple domain experts who would need to design and analyze such complex system, there is a need for efficient tools and facilities that enable *real-time collaboration* between these participating actors. In addition, as these domain experts work on business and sensitive processes who may want to keep their work *secure* and *private*, these collaboration tools must have a secure private storage for all collaborators.

Large-scale smart grid simulation execution is a highly compute- and data-intensive activity. Thus, a large pool of computing resources are required to support *high performance computing* of these simulations.

To address the above requirements – *co-simulation, real-time collaboration, security, privacy, and high performance computing*, we present, in this chapter, a design studio for collaborative modeling and co-simulation of mixed energy electrical systems.

V.2 Background

In this section, we will briefly cover the four fundamental technologies that enable our design studio. These include: (i) the co-simulation platform, (ii) the collaboration platform, (iii) the collaborative modeling web-bench, and (iv) simulation executions in the cloud.

V.2.1 Co-Simulation Platform

Owing to the lack of simulation tools' ability to cover multi-domain simulation characteristics, co-simulations offer an excellent alternative to analyze and simulate such multi-domain simulations. Co-simulation approach integrates different domain-specific simulators to provide a cohesive platform for carrying out studies for scenarios such as the mixed energy systems simulation for smart grids. *High-level architecture* (HLA) [2] provides a standardized method to integrate different simulators and execute them as a co-simulation. HLA provides information, synchronization, and coordination services among participating simulators. In HLA terminology, each of the participating simulator is called a federate. Different federates communicate with each-other by exchanging data according to their described publish and subscribe relationships. These different federates are time-synchronized – a crucial service for a distributed simulation. C2WT [85], developed at Vanderbilt University, is a heterogeneous simulation integration framework. It provides a model-based integration technology for rapid synthesis of distributed co-simulations such as those required for multi-domain simulations of cyber-physical systems (CPS). C2WT relies on the HLA standard and utilizes its open-source implementation (or Run-Time Infrastructure (RTI)) called Portico [18]. In this work, we are building on top of the C2WT framework to support cloud-scale simulations of mixed energy electrical systems in the context of smart grids.

V.2.2 Collaboration Platform

The Cyber-Physical Systems Virtual Organization (CPS-VO) [6] is a collaborative, web-based portal developed to promote interaction between academia, government, and industry across multiple disciplines in the burgeoning field of CPS. CPS-VO provides facilities for enabling repeatable, verifiable, shareable experiments and results to the users. CPS-VO supports multi-user collaboration communities with additional

experimentation cloud, tools configuration, and a testing framework, which enables rapid development of collaborative design solutions.

To support this, the CPS-VO has a three key elements: tool libraries, integrated tools, and design studios. Tool libraries are searchable repositories of available software categorized by multiple taxonomies. Integrated tools are software solutions that are embedded within the CPS-VO and ready to be utilized without having to setup a server, download, install, or configure anything. Embedded tools run in the CPS-VO cloud in order to retain the elasticity to accommodate changing demand while maintaining the security and stability of the CPS-VO. These capabilities of the CPS-VO are actively being utilized for CPS education. As more tools are being added to the CPS-VO, these elements are being increasingly utilized by a diverse set of communities.

For our purposes, we will be employing each of these features – listing the completed tool in the library, providing a design studio interface for designing multi-domain co-simulation experiments, as well as integrating a tool for executing these experiments. Results from the experiments will be retrieved from the cloud by the CPS-VO and securely made available to the user, with an ability to download or delete as needed.

V.2.3 Collaborative Modeling Web-Bench

Modeling is an important phase in the design of an experiment. For co-simulation experiments, modeling enables domain experts to design parameterized simulation models for studying how they perform when they interact with other simulation models participating in the co-simulation.

For collaborative modeling of co-simulation experiments, a modeling environment should support the following: (1) To enable multiple experts to collaborate and participate in building the simulation models, the platform should support collaborative

modeling. (2) The modeling environment should be able to support real-time editing and synchronization of simulation models across different participating domain experts. (3) Modeling environment should provide intuitive visual interface so that it lowers the entry of barrier to utilizing the new environment. This enable users to focus more on developing models rather spending time in learning a new toolsuite. (4) Provide tools for checking correctness of models and flagging constraint violations during designing of large-scale multi-model simulation experiments.

Taking these considerations into account, we selected the WebGME [106] modeling environment developed at Vanderbilt University. WebGME provides a web-based design and modeling environment. WebGME enable users to leverage model driven engineering techniques (MDE) [134] to develop large-scale software systems [35]. It provides facilities such as ability to create a visual domain specific modeling languages (DSML) using metamodeling. It allows creating model interpreters that are linked with the metamodels. Model interpreters enable automated software synthesis in the form of code artifacts and configurations, which are used by the domain experts to write simulation models and business logic to be embedded in the simulation models. WebGME supports checking model correctness and ensuring its conformity to set of constraints – a crucial while designing large-scale simulations. Visual notifications are shown for design time violations.

V.2.4 Cloud-Hosted Experimentation Platform

Large-scale smart grid simulation models are highly computation- and data-intensive. Thus, simulation models execution can benefit from the large resource

pool provided by the cloud computing model. Cloud computing provides an on-demand access to these compute resources for running simulation execution experiments. Cloud computing has been leveraged for running large-scale simulation execution [29, 99, 118] in understanding and studying architectures for building smart-city scale distributed systems.

Despite the advancement in the cloud computing systems, the research community is faced with numerous challenges in moving their simulation models to the cloud computing environment. Executing in the cloud computing environment needs understanding of the cloud-oriented configuration and deployment tools [44]. Insufficient expertise in these tools can lead to performance degradation of the executing processes [137]. Another challenge in using cloud computing is the difficulty in migrating simulation execution models from the desktop or laptop based execution platform to the distributed and scalable cloud platform environment. Another important consideration when running execution in the cloud environments is to avoid getting tied to a single cloud provider or what is called as 'vendor lock-in'.

To address these challenges related to the deployment of simulation executions in the cloud environment, we leverage open-source technologies such as Openstack cloud hosting [16] and linux container based Docker technology [7] to host simulations. We will cover more details in the next section.

V.3 Design and Implementation of Design Studio

In this section we will cover the design and implementation of the collaborative platform for modeling and simulation of the mixed energy smart grid simulations.

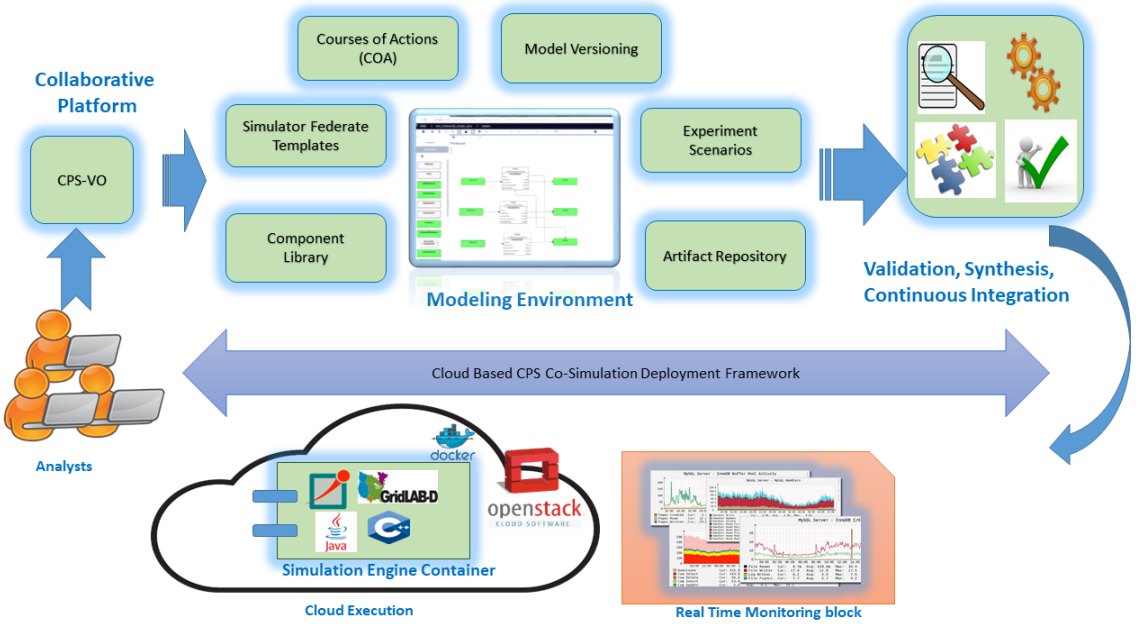

Figure 37: Overview of the Cloud based Modeling and Co-simulation of Mixed Electrical Energy Systems

V.3.1 Design And Architecture

Many building blocks are required to support collaborative modeling, simulation, and execution of the co-simulation experiments. Figure 37 showcases the main components of the design studio platform. These components are described in detail below.

V.3.1.1 CPS-VO

As discussed in V.2.2, the CPS-VO provides an entry portal to the users of the design studio. It provides users and user groups management functionality. This provides features such as user authentication and permission management to enable secure and private collaboration among users within a group or community. User access to the community portal is managed through a highly customized Drupal PHP based web portal. Access to the WebGME based modeling environment and

the experimentation portal is channeled via the CPS-VO portal. When transitioning from simulation modeling to running an experiment, it facilitates access to the run-time cloud infrastructure. Currently, the CPS-VO provides a naive experiment scheduling policy by restricting the number of concurrent experiments to the total available execution computing nodes. Experiment results are automatically retrieved by the CPS-VO after the experiment has finished, and are stored and made available according to the privacy settings the user has selected, combined with group settings.

V.3.1.2 Simulator Federate Templates

This component provides access to various types of federates which are simulator specific. These includes C++, Java, Gridlab-D, and OMNET++ simulation engines. Users can build scenarios utilizing these simulation engines to construct large-scale simulations. Using these templates, one can construct an integration model of the simulated scenario. This integration model represents different federate type entities participating in the simulation. The integration model also covers any interactions, shared objects that may be exchanged between participating federates.

V.3.1.3 Courses Of Action Models

To conduct scenario-based experimentation and conducting what-if analysis [117], we support a modeling construct called as Courses-of-Action (COA). COAs are utilized to create various what-if analysis models and to execute the corresponding alternative scenarios. COAs act as an orchestrator of the time-coordinated execution of the running simulations. These COAs are scenario models that are created using several atomic elements such as: *ACTION* – that injects an interaction into the running simulation, *OUTCOME* – that waits for an interaction of the specified type to be generated in the running simulation, *FORK* – that start multiple branches in a scenario to start in parallel, and *DUR* – that, when encountered in a COA execution,

makes a running simulation wait for the specified duration. A COA model is created as a workflow like Directed Acyclic Graph (DAG) by connecting the above-mentioned atomic elements with directed edges. Detailed description of many other supported COA elements can be found in [99].

V.3.1.4 Experiment Scenarios Models

Once the simulation integration model is designed with the constituent simulation federates, the data model for the data exchange among federates, and the objects that capture various actors in the co-simulation, the next step involves creating the experimental scenario models. Experimental model enables creating scenarios that comprise either some or all the federates from the integration model created in the previous step. As such, for a given co-simulation scenario we could have more than one experiment model.

V.3.1.5 Software Synthesis

To facilitate rapid development of the co-simulation application, design studio features a code generation and synthesis module. This module enables synthesis of the *integration code* – software modules that bridge target simulators with the underlying HLA RTI. This module provides two key benefits. Firstly, it ensures the 'correct by construction' principle when generating the large boilerplate code required for such complex co-simulation RTI, thereby avoiding errors that occur with manually written code. Secondly, it lowers barrier to entry to the development of co-simulation application, whereby the domain expert can focus on writing simulation models and not worry about the complexity to work with the underlying RTI.

Apart from above software synthesis components, code generators also produce various experiment specific configurations, which are specific to the experiment model

discussed earlier. These includes COA models, various federate configuration specific to the experiment model using JSON [11] files, and several initialization scripts.

V.3.1.6 Artifact Repository

All the artifacts that are part of the co-simulations are stored in a secured centralized repository with user access control. These artifacts includes auto-generated software codes, user supplied application programs, simulator specific models, WebGME models, and various configuration files associated with the co-simulation experiment. Artifact repository provides storage facility to store these co-simulation artifacts. The repository also features version control mechanism, such that the artifacts can be associated for a given version number and can optionally include labels such as development, beta, and production ready tags.

V.3.1.7 Continuous Integration

Continuous integration is widely used technique in the software development process for creating automated software builds that run various tests that validate whether the software compiles and builds successfully. It can also include unit tests to ensure that the newly developed simulation module meets certain functional requirements as specified in the unit tests. Continuous integration enables automatic compilation and building of the source code generated by the software synthesis module and storage of the compiled artifacts to the artifact repository discussed earlier. Currently, we are leveraging the Jenkins [12] build systems to trigger automatic builds of the simulation software.

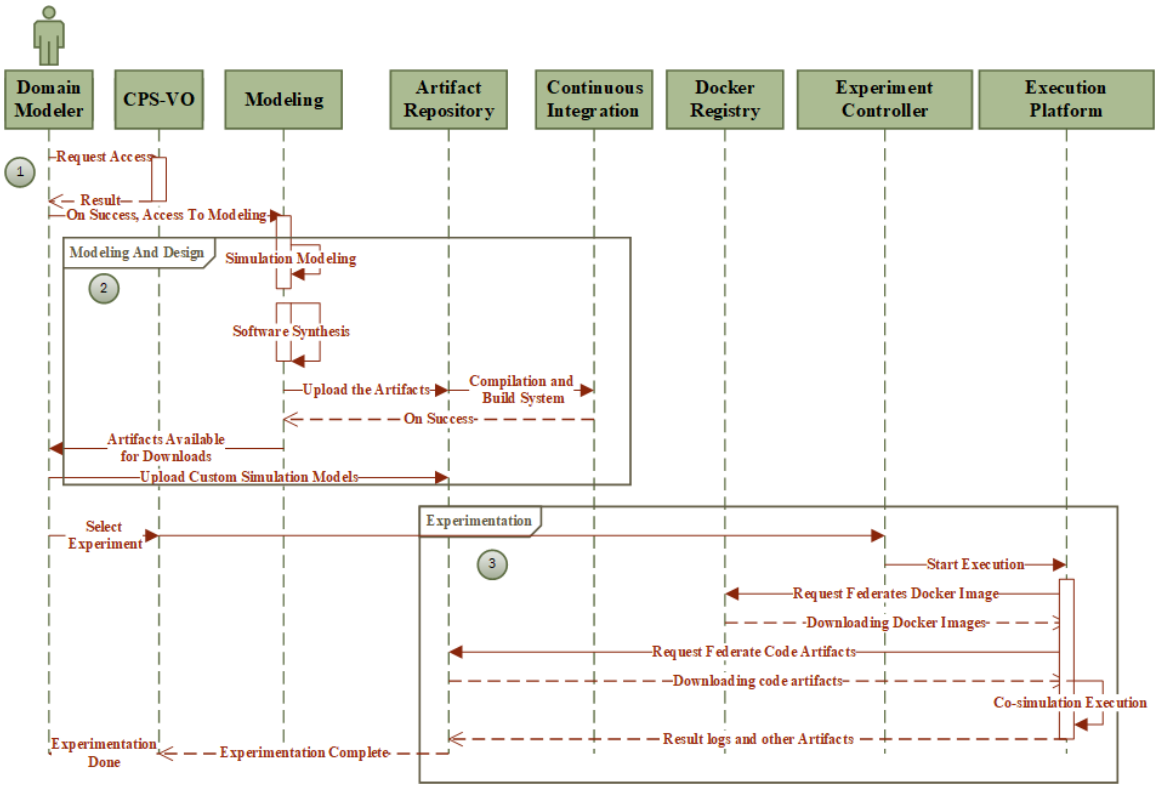

Figure 38: Sequence diagram showcasing modeling and experimentation activity in design studio

V.3.1.8 Experimentation Runtime

To support large-scale distributed simulation, which may exhibit different compute, input/output, and/or network intensive workload characteristics, a cloud environment provides a better execution infrastructure to support these requirements. One of the requirements to support cross-platform executions is to enable running simulations across heterogeneous run-time platforms. Docker container technology [7] is utilized to meet this requirement. To enable running simulators inside docker containers, we first have to port the simulators to the docker run-time image. Once this image is created, it is available to run simulator-specific models. Currently, our design studio makes the C++, Java, OMNeT++, and CPNTools docker images available for the domain modelers.

V.3.1.9 Monitoring and Visualization

Live monitoring of co-simulation experiments provides insights into how the simulation experiment is running in the cloud environment. We are leveraging Grafana [9]) based visualization dashboards to depict important events that arise in the running simulation. Grafana dashboards also show the runtime system resource utilization of the individual federates comprising a co-simulation experiment run. Monitoring facility can thus be utilized for profiling simulation runs, and could be used in the future for making dynamic resource management decisions for running the distributed co-simulation experiments.

V.3.2 Modeling and Experimentation Workflow

In this section we will cover the modeling and experimentation workflow in the design studio framework. Figure 38 shows the two activities which the domain modeler performs in the design studio framework: co-simulation modeling and running co-simulations. Next, we will refer to the circle numbers as they are shown in the sequence diagram. In ①, the domain modeler first enters the design studio by authenticating with the CPS-VO portal. Once authenticated, the domain modeler has access to the WebGME modeling application. Next in step ②, the domain expert can first build the co-simulation design model and configure various experiment scenarios. Once the modeling activity is completed, software synthesis module generates simulation specific code artifacts. These artifacts are available to the modeler to download and update them with application-specific code and/or models. In addition, the software artifacts generated are passed to the continuous build and integration system for running versioned builds. The modeler can also upload custom simulation artifacts and updated software artifacts to the artifact repository, which can then be utilized accordingly for the co-simulation. Further, in step ③, the user than selects one of the configured experiment model from the previous step for executing

on the experimentation platform. Once the run experimentation option is selected, the experiment controller then selects an appropriate runtime platform server from the available cloud infrastructure, and starts the experimental run sequence. During the startup of the experiment, appropriate Docker federate images, as required by the experiment, are downloaded from the docker registry hosted within the CPS-VO environment. Furthermore, the experiment specific federate code artifacts are downloaded on the execution server from the artifact repository. Once the required dependencies are downloaded, the co-simulation execution can begin. The simulation proceeds and simulates the experiment scenarios. Once the simulation criteria is met, which is set by the user based on either the amount of simulation time to execute or a specific simulation objective is met, the simulation execution stops. The simulation gets the execution trace and the generated results and logs are then uploaded back to the CPS-VO for the offline analysis.

V.4 Conclusions And Future Work

This chapter presents a cloud based secure, collaborative modeling and co-simulation platform to support mixed electrical energy systems simulation for smart grid operations. We have described various building blocks of the integrated design studio. We leverage the WebGME tool to provide a the web-based modeling environment and the CPS-VO to provide the central collaboration platform. Using continuous integration technology the simulation artifacts are automatically built and ready to be deployed to the cloud execution platform. Docker technology provides a cross-platform sandboxed execution environment for the co-simulations which can be executed on the elastic cloud computing resource.

There is a general lack of integrated toolsuites that can provide collaborative modeling and co-simulations facilities for complex applications such as mixed energy electrical systems. The presented design studio can enable various stakeholders in the

smart grid environment to effectively collaborate with multi-user modeling of experiments, and to design and build resilient and high-performance smart grid systems. In future, we plan on adding support for efficient deployment and configuration management for the distributed simulations in the cloud environment [33], save and restore of the running simulations, and tool support for wider range of simulators such as EnergyPlus [64], and DIgSILENT [83].

CHAPTER VI

ADDRESSING PERFORMANCE ISSUES IN DISTRIBUTED CO-SIMULATIONS

VI.1 Introduction

Cyber-physical systems (CPS) such as smart city, smart manufacturing, and transactive energy systems must make time-sensitive control decisions to ensure safe operations of such complex systems. However, such CPS are an amalgamation of multiple dynamic systems such as transportation systems, vehicle dynamics, control systems, and power systems with different interconnected networks of different scales and properties. The assurance that such complex systems are safe and trustworthy requires simulation capabilities that can rapidly integrate tools from multiple domains in different configurations. Co-simulation is an attractive option for interlinking such multiple simulators to simulate higher-level, complex system behaviors.

The IEEE 1516-2010 High Level Architecture (HLA) defines the standardized set of services offered to a process in a distributed co-simulation [20]. A co-simulation in HLA comprises a group of individual simulators called federates that are grouped into a logical entity called a federation. Each federate could have diverse computation and networking resource requirements, which need to be taken into account when deciding how to allocate resources to the simulators when the federation is deployed to cloud-fog-edge computation environments. Based on this resource allocation, the wall-clock execution time required for the computation step of each federate can vary. The variation in execution times can result in some federates waiting on others before they can proceed to the stage of computation. These federates that take more wall-clock time for completing their computing steps (resulting in low performing federates), can increase the overall completion time (or *makespan*) of the entire simulation. This

can potentially violate the simulation completion deadline. Thus, resource allocation and resource configuration selections are very important for the overall performance of distributed simulations.

Although co-simulations have traditionally been hosted in high performance compute (HPC) clusters, there is an increasing trend towards adopting Cloud Computing for simulation jobs because of the many benefits it has to offer. It is in this context that the Docker container run-time platform [109] provides a solution for running federates across different computation platforms by providing a unified packaging of the simulation code-base with its software dependencies. But as shown in recent research [145], there are a number of operationalization challenges that need to be considered for running simulations in cloud environments such as performance aware resource assignments. Furthermore, there are a several infrastructure-related accidental complexities that need to be considered for running these distributed simulations on such execution platforms.

To address these challenges, this work introduces a performance profiling and simulation run-time optimization and resource configuration platform called EXPPO - (***EX****ecution* ***P****erformance* ***P****rofiling and* ***O****ptimization for CPS Co-simulation-as-a-Service)*). EXPPO uses distributed tracing to assess federate level performance for each computational step [105]. These performance characteristics are used at runtime to determine whether changes to the resource allocation of the federation could enable shorter makespan for the simulation run. To enable distributed tracing, EXPPO utilizes the *opentracing* [123] specification for measuring and monitoring the wall-clock time spent in each computational step of the simulation. To shield the developers from accidental complexities while embedding the tracing source-code in the simulation application logic, EXPPO leverages generative aspect of Model Driven Engineering (MDE) to auto-generate source-code snippets and configuration files for the simulation run. To configure the resource allocation for the individual federates,

EXPPO uses the tracing information to build a resource-performance model to solve an optimization problem resulting in resource allocation having lower makespan and cost for the co-simulation.

The main contributions of EXPPO are:

1. Evidence that the default resource configuration for a federation has scenarios with longer wall clock execution per computational step, causing a longer makespan for the co-simulation execution;

2. Demonstration of a methodology to automatically generate tracing probes inside the source-code of the simulation logic, which is required for simulation profiling at distributed scale;

3. Development of a new resource recommendation engine which uses an optimization algorithm for finding the resource configuration for a federation that minimizes its overall makespan and cost;

4. Experimental validation of the proposed scheme showing the benefits of EXPPO on the performance of distributed simulation execution.

The rest of the chapter is organized as follows. Section VI.2 provides a motivating use-case for EXPPO and enumerates its key requirements. Section VI.3 provides an overview of the EXPPO tool. Section VI.4 presents the key EXPPO components and its resource configuration and optimization algorithms. EXPPO's cloud architecture and co-simulation framework are described in Section VI.5. The experiment evaluation results are described in Section VI.6, related works described in Section VI.7 and Section VI.8 concludes the chapter.

VI.2 Motivation and Solution Requirements

The computation pattern of many time-stepped co-simulations follows the Bulk Synchronous Parallel (BSP) model [152]. In this model, every participating simulation

completes its computation for a given time step, waits for the other participating simulations to complete their computations, exchanges new state information with its peers, and only then proceeds to the next time step. Thus, if one simulation takes more time to execute, the other simulations are forced to wait for it and remain idle. This is illustrated in Figure 39 which shows an HLA federation with three federates: Federate 1, Federate 2, and Federate 3. Each federate has same computation task repeated for every following time step. Federate 1 performs certain computation which takes the longest execution time for a given simulation time step, compared to the other two federates. Thus, the other two federates are waiting for Federate 1 to complete before moving to the next computation step. This causes increase in the makespan of the entire simulation, thereby decreasing the performance of the simulation execution.

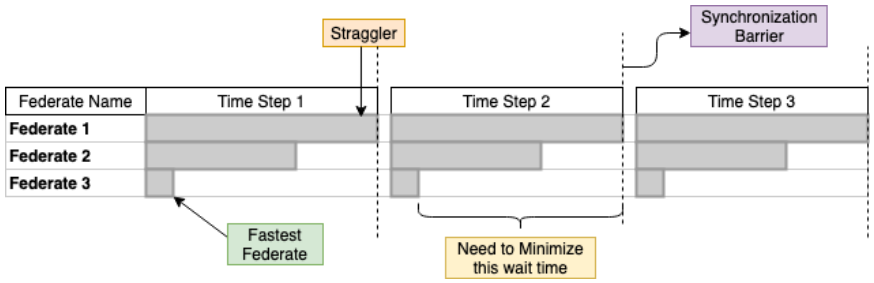

Figure 39: Performance visualization of an example federation with three federates.

Recently, co-simulations have been deployed using container management solutions [130] [28], such as Docker Swarm [71], Kubernetes [100], Mesos [87]. However, these solutions use queue-based scheduling, wherein the containers are allocated to machines one at a time. Hence, there could be instances where there may not be enough resources available in the cluster to schedule all the federates of the federation. However, due to the scheduler's lack of federation aware container affinity knowledge, a few federates may still get deployed until the resources are available

and some of the federates will remain in the scheduler queue. This will cause the entire simulation to stall, since all the federates are required to participate in the simulation to progress. For instance, from Figure 39, if there are resources to deploy Federate 1 and Federate 3, and not sufficient resources to deploy Federate 2, the job scheduler should not deploy any federates and should wait until more resources are available. Hence, there is a need for Bag-of-task scheduling mechanism for deploying these federates on the cloud computing environments.

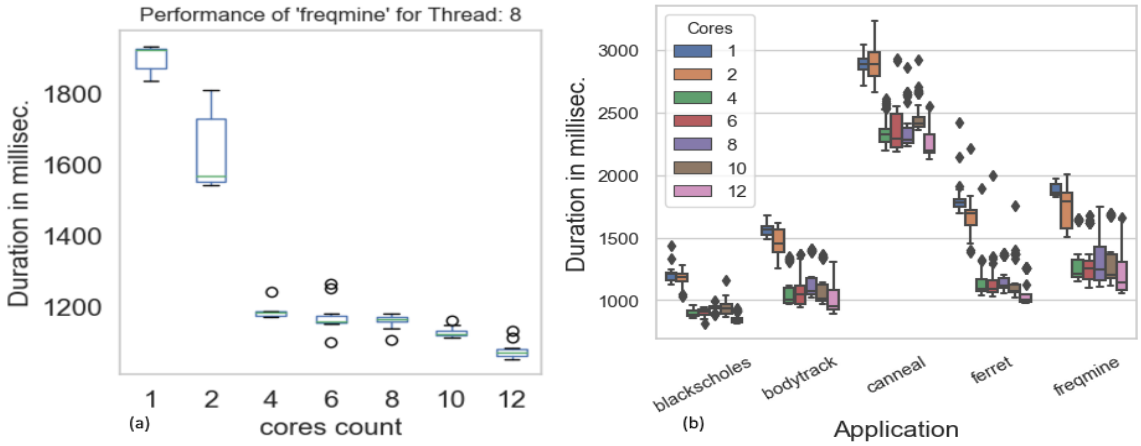

Figure 40: (a) Performance of the federate running the PARSEC Freqmine application using 8 threads for different resource configuration selections. The performance improves when assigned more cores. (b) Performance of five PARSEC benchmark applications for different resource configuration selections. The performance improves when assigned more cores.

The resource configuration also plays an important role in the execution time of the federates. Figure 40(a) depicts the performance of a federate when assigned different numbers of cores for executing a computation step. This federate is running a Freqmine application from the PARSEC benchmark [46] as its computation task. It can be seen that the execution time follows a non-increasing trend with the increasing amount of resources assigned to the federate. Thus, there is a potential for minimizing the wait time by appropriately configuring the resources assigned to the federates.

Minimizing the wait time can be done either by providing the highest resource configuration for all the federates, or finding a resource configuration that considers the cost of assigning the resources to the federates. However, deciding what resource configuration to select for a given computation is a non-trivial task for a simulation developer who may not have the domain expertise of configuring and running applications in cloud computing environments. Performance profiling of the federates can help in understanding the relation between the resource assignment and the execution performance. However, these simulations might be deployed across different physical and/or virtual host environments when running in cloud computing platforms; performance profiling for such distributed simulations can be very challenging.

Building on the above use-case, below are the four key requirements that must be satisfied by EXPPO:

- **Requirement R1.** *Conduct distributed performance tracing of the federates:* To understand the bottlenecks in federation performance, there is a need for logging and gathering execution performance traces of the federates. The tracing infrastructure needs to handle federates which are distributed across multiple physical hosts. Hence, the tracing infrastructure should be able to correlate traces from different federates of a federation.

- **Requirement R2.** *Reduce accidental complexity in provisioning software probes:* Requiring the developer to manually write source code for performance tracing for the federation can result in accidental complexities and errors which need to be minimized. The developer needs to understand the tracing software and write code which adheres to the tracing software requirements. This is tedious for the developer, who now apart from writing the simulation logic must also setup and configure the tracing infrastructure. EXPPO should reduce the manual configuration of the tracing information, thereby reducing repeated effort by the developer.

- **Requirement R3.** *Recommend resource configuration to minimize the co-simulation makespan and cost:* It can be challenging for an end user to determine the resource requirements for a federation because each federate can be configured differently. A bad resource configuration can have inadvertent effect on the completion time of the simulation. Also, choosing configuration incurs cost. Hence, the resource configuration must be chosen such it satisfies users QoS and cost factors.

- **Requirement R4.** *Provide a gang-scheduling algorithm for executing simulations:* The runtime platform should support deployment of multiple federate using bag-of-tasks scheduling (also referred as gang scheduling) algorithm.

VI.3 Overview of EXPPO

Figure 41 introduces the workflow and the main components of EXPPO.

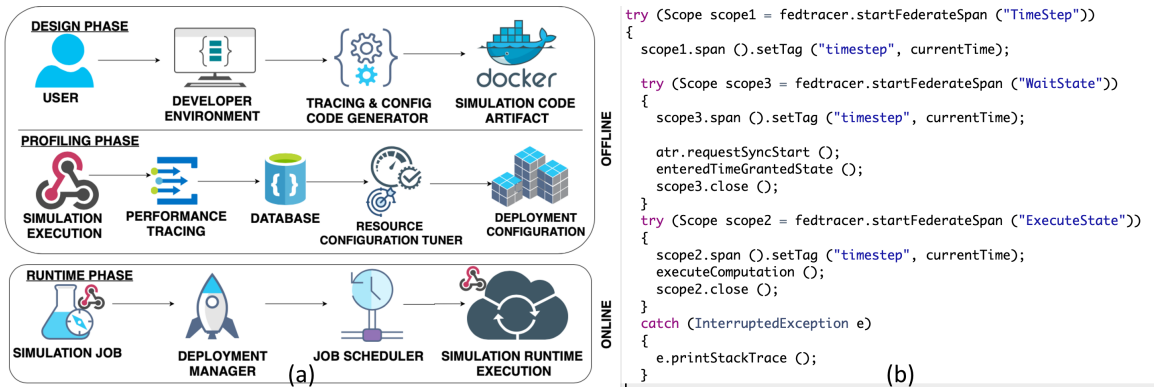

Figure 41: (a) Workflow of EXPPO illustrating the connections between different components of the system. (b) Profiling code snippet in Java language generated leveraging the MDE techniques.

The *design phase* requires the developer to model the federation using the federation development toolkit. This toolkit is based on the Web-based Generic Modeling Environment (WebGME) [106]. To measure the execution time of a federate in a time

step, the simulator needs to embed a tracing code initializer and logger to record the execution time completion of the computation step. The tracing initialization code and the tracer configuration files are auto-generated by the custom WebGME model interpreters. Once the required code is generated and the user has implemented the necessary simulation logic, the federate is compiled into an executable image, which is then packaged inside a Docker container.

During the *profiling phase*, each federate is executed and profiled under different resource configurations. The execution time for each federate is logged using the tracing information, and this tracing information is stored in a centralized database. The resource configuration tuner uses the recorded logs together with user provided objectives to optimize the resource configuration for each federate. Finally, the optimized resource configuration is used to configure the co-simulation deployment accordingly.

During the *runtime phase*, the simulation job information for the federates in the co-simulation is submitted to the deployment manager. The job information includes the name of the federate Docker image, resource configuration requirements, etc. The deployment manager submits the scheduling information to the job scheduler, which handles the execution of the jobs on the co-simulation runtime execution platform.

VI.4 Design Elements of EXPPO

EXPPO allows users to design and deploy co-simulations on distributed compute infrastructures supported by Docker-based virtualization. Its generative capabilities simplify the auto-generation of performance monitoring instrumentation, configuration and probing for the different federates. Its resource configuration tuner optimizes resource allocations to federates to lower the makespan and execution cost for the federation execution. Its runtime platform supports parallel execution of different federations on its shared compute infrastructure. This section details each of the EXPPO components.

VI.4.1 Performance Profiling of Federates

Understanding the performance characteristics of the co-simulation execution is of paramount importance when deciding how to run them in runtime execution environments. Individual federates have different resource needs, and their execution performance will vary depending on how the resources are assigned. Thus, understanding the performance profiles of each federate is critical to the problem of optimizing the resource allocation and thereby lowering the makespan and execution cost of the federation execution. EXPPO leverages distributed tracing to assist in the logging of time stamps of distributed events generated in the federation (Requirement **R1**). It leverages Opentracing instrumentation [123] to track execution time spent during each computation time step of the federate. This information in then logged into a timeseries database for conducting performance analysis. It leverages model driven engineering technologies such as Domain Specific Modeling Language (DSML), code generators and model interpreters to synthesize performance profiling software artifacts which can be used for performance profiling of federates (Requirement **R2**). An example code snippet is shown in Figure 41 (b).

VI.4.2 Federation Resource Configuration Optimization

When running a federate in a Docker container, cloud providers usually have multiple resource configurations from which the user can select for their application. However, without analyzing the resource dependency of the application, it may be challenging for the user to select the resource configuration that meets the application's quality of service (QoS) requirement while at the same time minimizing the execution cost in terms of the cost of resources utilized. Figure 40(b) shows how the different resource configuration impacts the execution time of the federate which is running applications from PARSEC benchmark. Furthermore, in a co-simulation,

the resource selection becomes critical as every federate's execution time will contribute to the co-simulation's performance. To address this issue (Requirement **R3**), EXPPO provides a resource configuration recommendation system that selects the resource assignment which optimizes the application's QoS performance and user's budget requirements.

Consider the following optimization problem: Suppose the system has a set of homogeneous machines, each with k processing cores. Let $F = \{f_1, f_2, \ldots, f_n\}$ denote a federation (co-simulation) that consists of a set of n federates. Given a resource assignment $R = [r_1, r_2, \ldots, r_n]$ to each federate, the execution time of federate $f_j \in F$ can be expressed as $t_j(r_j)$ when assigned r_j cores. For performance reasons, assume a federate cannot be split among two or more machines, so we have $1 \leq r_j \leq k$. The makespan M for every computation step for the entire federation F is dictated by the slowest running federate (i.e., the straggler), and is defined as $M = \max_j t_j(r_j)$. The execution cost C is given by the total resource used by all the federates over the makespan duration. Since the servers will be turned ON until the slowest federate is done executing, the cost is defined as $C = M \cdot \sum_j r_j$. The resource configuration recommender needs to find a resource assignment R^* that minimizes $G = \alpha M + \beta C = M(\alpha + \beta \sum_j r_j)$, where α and β denote the user-defined weights to the application's QoS and the execution cost, respectively.

Two additional assumptions are made to solve this optimization problem: (1) federate execution time does not increase with the amount of resources (number of cores) assigned, i.e., $r_j \leq r'_j$ implies $t_j(r_j) \geq t_j(r'_j)$; (2) federate execution cost does not decrease with the amount of resources assigned, i.e., $r_j \leq r'_j$ implies $c_j(r_j) \leq c_j(r'_j)$. These are realistic assumptions as many practical applications are known to have *monotonically increasing and concave* speedup functions [39, 86], such as those that follow the Amdahl's Law [22]. As such, the optimization problem can be solved by examining all possible makespan values while guaranteeing the smallest

cost. Algorithm 2 presents the pseudocode of this solution with a time complexity of $O(n \log n)$ by maintaining a priority queue for all jobs. Similar approaches can be applied to finding the minimum makespan subject to a cost budget or minimizing the cost for a target makespan.

Algorithm 2: Resource Configuration Tuner

Input : Execution time $t_j(r)$ for each federate f_j in federation F when allocated different amounts of resources r, where $1 \leq r \leq k$.

Output: A resource assignment $R^* = [r_1, r_2, \ldots, r_n]$ for each federate in the federation that minimizes a linear combination of makespan and cost.

```
Initialize r_j ← 1 for all 1 ≤ j ≤ n;
Compute G ← (max_j t_j(r_j)) · (α + β Σ_j r_j);
R* ← [r_1, r_2, ..., r_n] and G* ← G;
while Σ_j r_j < nk do
    j ← Index of a federate with longest execution time;
    if r_j = k then
        break;
    else
        Increment r_j to the next higher profiled resource amount;
        Update G ← (max_j t_j(r_j)) · (α + β Σ_j r_j);
        if G < G* then
            R* ← [r_1, r_2, ..., r_n] and G* ← G;
```

VI.4.3 Federation Machine Scheduling Heuristics

A custom scheduler is required to deploy all the federates in the federation to their respective distributed computing environments. Since the federates cannot run independent of the federation, the scheduling scheme must simultaneously run all of the federates of the federation (Requirement **R4**). This is referred to as *Gang scheduling* or *Bag-of-tasks scheduling* in the literature. To achieve this, EXPPO supports two heuristics to simultaneously schedule the federates on a fixed number m of available machines while utilizing the resource configuration results obtained from

Section VI.4.2. These approaches handle the case where some of the machines are loaded with other compute tasks unrelated to the federation.

The first heuristic is inspired by the First-Fit Decreasing (FFD) algorithm for bin packing and is described in Algorithm 3. The heuristic first sorts all the federates in decreasing order of resource assignment and then tentatively allocates each one of them in order onto the first available machine. If all federates in the federation can be successfully allocated, then the schedule is finalized; otherwise, the entire federation will be temporarily put in a waiting queue to be scheduled later. The time complexity of the heuristic is $O(n(\log n + m))$. The other heuristic is based on the Best-Fit Decreasing (BFD) algorithm that works similarly to FFD, except that it finds, for each federate, a best-fitting machine (i.e., with the least remaining resource after hosting the federate). Note that since finding the optimal schedule (or bin packing) is an NP-complete problem, these heuristics may not always find a feasible allocation for a federation even if one exists. However, once an allocation has been found, it is guaranteed to produce the optimal makespan and cost for the federation by using the resource configuration from Section VI.4.2.

Algorithm 3: First Fit Decreasing (FFD)

Input : Resource assignment $R = [r_1, r_2, ..., r_n]$ for all federates in a federation. Current available resource $A = [a_1, a_2, ..., a_m]$ of all m machines in the system.

Output: Machine allocation $L = [\ell_1, \ell_2, \ldots, \ell_n]$ of all federates in the system.

Sort all resource assignments in decreasing order, i.e., $r_1 \geq r_2 \geq \cdots \geq r_n$;
Initialize $\ell_j \leftarrow 0, \quad \forall 1 \leq j \leq n$;
for $j = 1, 2, \ldots, n$ **do**
    $fit \leftarrow false$;
    for $i = 1, 2, \ldots, m$ **do**
        if $r_j \leq a_i$ **then**
            Update $a_i \leftarrow a_i - r_j$;
            Set $\ell_j \leftarrow i$;
            $fit \leftarrow true$;
            break;
    if $fit = false$ **then**
        `// revert allocations done so far`
        for $k = 1, 2, \ldots, j-1$ **do**
            $i \leftarrow \ell_k$;
            $a_i \leftarrow a_i + r_k$;
            $\ell_k \leftarrow 0$;
        break;

VI.5 Co-simulation as a Service Middleware Architecture

Figure 42 shows the different components and workflow of our co-simulation framework. It is called the Co-simulation-as-a-Service (CAAS) middleware because it enables users to automatically deploy groups of service-based applications to a cloud environment without any concern of resource allocation, application lifecycle monitoring, and cluster management. The functionality of each component is described below.

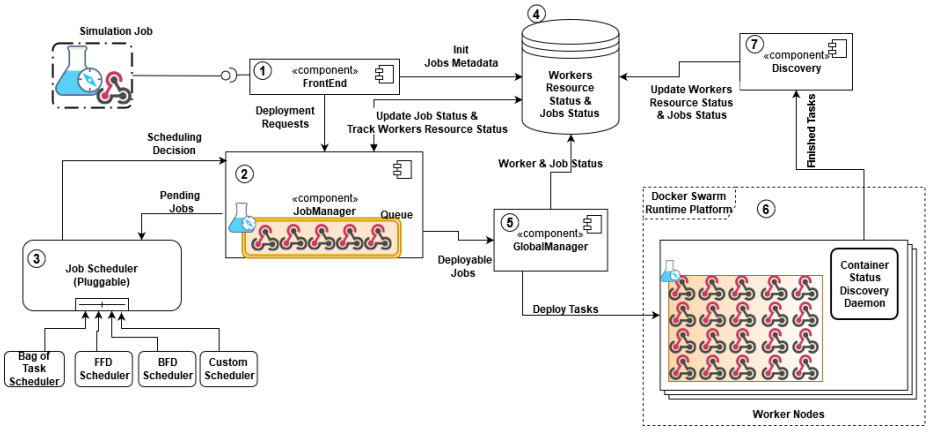

Figure 42: Co-simulation-as-a-Service

In ①, the FrontEnd component allows a user to submit a simulation job descriptor in JavaScript Object Notation (JSON) using a Representational State Transfer (REST) Application Programming Interface (API). Each simulation job contains a list of federates, and each federate is run on an individual Docker container. The simulation job descriptor also includes meta-information for each federate, for example, resources required, running status, container image details, etc. The FrontEnd creates a record in a database ④ for each incoming job, then relays the job identifier to JobManager ② which handles resource management. Instead of deploying jobs immediately, the JobManager stores received jobs in a local queue and consults the database about the latest status of cluster resources. It then periodically

transfers the information of pending jobs and available resources to JobScheduler ③, which is a pluggable component that implements multiple scheduling algorithms. The JobScheduler either replies with a scheduling decision if the submitted jobs are deployable, or returns a KeepWaiting signal to notify the JobManager that resources are insufficient. The JobManager then forwards deployable jobs to GlobalManager ⑤, synchronizes job status with the database, and deletes the deployed jobs from its queue. The GlobalManager is responsible for managing participants of the Docker Swarm Runtime Platform ⑥. It launches the master node of the Docker Swarm cluster and accepts registration requests sent from worker nodes. Every joined Worker Node runs a CAAS-Worker daemon that is used to receive and perform commands sent from the GlobalManager. The GlobalManager parses received deployment requests and spawns containers in specific Worker Nodes. Additionally, the Worker Nodes employs a CAAS-Discovery daemon to track the status of containers, and reports a StatusChanged signal to the Discovery component ⑦ when it detects that a task is completed.

VI.6 Experimental Evaluation

VI.6.1 Experimental Setup

EXPPO is validated using seven homogeneous compute servers with configuration of 12 core 2.1 GHz AMD Opteron central processing units, 32 GB memory, 500 GB disk space, and the Ubuntu 16.04 operating system. The runtime platform is based on Docker engine version 19.0.5 with swarm mode enabled. There is one client machine that submits simulation job requests. The front end, job manager, job scheduler and the global manager components are runing on a single shared compute server, and five compute worker servers are deployed for running the simulation jobs.

The simulation job consists of three federates each running a unique application from the PARSEC benchmark: freqmine, blackscholes and ferret. These applications

were chosen as they are realistic representations of real-world simulation tasks [46]. During the federation execution, each federate executes its application at every logical time step, for a total number of 100 logical time steps. The implementation of the co-simulation federation is done using Portico HLA [18]. The BFD scheduler was used in the experiments (as it was found to have better performance than FFD). The weights for the QoS and the cost are set to $\alpha = 1$ and $\beta = 0.5$.

VI.6.2 Experimental Results

EXPPO selected resource configuration of 4 cores for freqmine federate, 4 cores for ferret federate and 1 core for blackscholes federate. Two baseline approaches were used to compare the performance of resource configuration selection of EXPPO. In the first approach (*least configuration*), all the federates are assigned the lowest possible configuration of 1 core each. In the second approach (*max configuration*), all the federates are assigned the highest possible configuration of 10 cores each. The performance data is collected over 10 simulation jobs.

Figure 43(a) shows the cumulative distribution function (CDF) of the execution time of EXPPO compared to the other two approaches. The resource configurations selected by EXPPO for the federation had a $90th$ percentile execution time of around 230 seconds, which is signficantly better than the 320 seconds for the *least configuration*. Its performance was close to that of the *max configuration* ($90th$ percentile execution time of around 200 seconds), with the difference due to its lower resource allocation to conserve the cost.

Figure 43(b) shows the cost analysis of the three strategies. As can be seen, resource configuration selected by EXPPO incurs a larger cost than the *min configuration* due to the higher resource allocation to reduce execution time, but it has a substantially lower cost compared to the *max configuration*.

Overall, the results show that EXPPO is able to select resource configurations and

schedule the federates in such a way that minimizes the combination of execution time and cost of the simulation successfully.

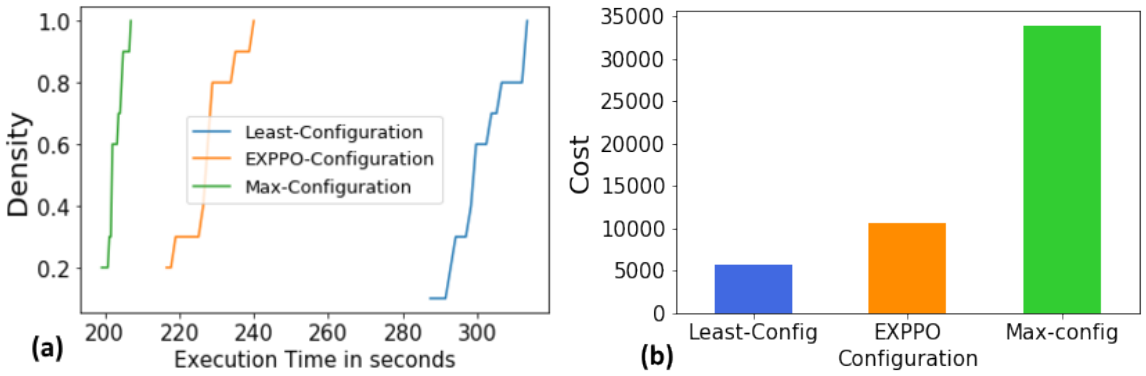

Figure 43: (a) Execution time for the EXPPO resource configuration compared to other strategies. (b) Cost for the EXPPO resource configuration compared to other strategies.

VI.7 Related Work

In [130], the authors presented Kubernetes based co-simulation execution platform for cloud computing environments. Similarly, [28] presented Docker swarm based co-simulation platform for running mixed electrical energy systems simulations. However, these platforms lack gang-scheduling based simulation deployment strategy.

Similarly, for resource recommendation, [157] presented data-driven approach for selecting best resource configuration VM from a set of VM configuration pool. In [135], cost-sensitive allocation of independent tasks is presented for cloud computing environment. However, these approaches are different than our approach as our objective is to select best configuration for a pool of BSP tasks and not just single task.

In [103], the authors proposed scientific workflows scheduling algorithm to minimize the execution time under budget constraints for deploying in cloud computing

environments. [79] proposed advancereservation scheduling strategy for MPI applications. Similarly, [154] presented approach for gang-scheduling of jobs by resource sharing among jobs of different resource needs, such as one being compute intensive and other of network or I/O intensive type. In [88], the authors present scheduling of multiple container workloads on shared cluster as a minimum cost flow problem (MCFP) constraint satisfaction problem. In [160], the authors proposed a locality based process placements for parallel and distributed simulation. The evaluation of the proposed framework was carried out using OMNET++ simulator.

Compared to the above prior works, EXPPO presents a resource recommendation engine which allows for selecting appropriate resources for the federates of the federation to meet the goals of make-span and cost minimization for the entire federation (BSP tasks) and not just a single task. Similarly, EXPPO also presents a gang scheduling scheme using heuristic bin packing techniques which allows for deployment of the co-simulation federation on Docker container platform thereby addressing limitation of one container at a time scheduling discussed earlier. Furthermore, EXPPO provides automatic code generation of distributed tracing probes for performance profiling of the federates which maybe deployed on a distributed infrastructure, relieving the developers from incurring accidental complexities in writing code for the profiling of federates.

VI.8 Conclusion

Resource allocation plays a critical role in co-simulation performance. However, the end user is not necessarily well-equipped to determine what resource allocations work best for their distributed, co-simulation jobs given the various resource configuration options (and associated costs) available from the cloud provider. To address these challenges, this paper presents EXPPO, which is a framework that provides CPS Co-simulation-as-a-Service (CaaS) for executing distributed co-simulations in

cloud computing environments. EXPPO provides performance profiling capabilities for federates which help in the understanding of the relationship between resource allocations and the simulation performance. Similarly, it addresses performance profiling challenges for a simulation job comprising heterogeneous computation tasks or federates deployed across distributed systems. Furthermore, EXPPO selects the resource configurations for these federates in a way that not only minimizes the makespan of the co-simulation, but also satisfies the cost budget of the user.

VI.8.1 Lessons Learned & Future Work

- Our current work assumes that the computations at each simulation step of a federate do not change and also takes almost the same wall clock execution time. However, this assumption restricts the generality of our approach to only a subset of application use cases. For hybrid simulations or simulations which follow non-linear dynamics, the execution times of the involved simulation steps may not be the same for each iteration. Hence for our future work, we would like to explore dynamic resource allocation for federates which may have different computations for different computation steps. Reinforcement learning approaches which can monitor the individual federates and dynamically adjust the allocations based on the computation variability or resource demands of the federate at different time steps offers a promising approach that remains to be explored.

- Further, in our current work, we have only considered workloads that are CPU bound. Hence, we assume constant communication costs during each computation step for all the federates. In future we would also like to consider the communication costs for simulations which may need to be constantly updating states or sending messages for triggering discrete events. This new requirement will play a factor during the placement of the federates on the worker nodes. It

may perhaps be a better placement for two communication intensive federates to be located on the same physical host rather getting deployed across different host machines.

- When applications are co-located in the cluster environments, the effects of noisy-neighbors can affect application by incurring performance interference. Thus, for future work we would like to consider the effect of noisy-neighbors for resource scheduling and simulation placement in the cluster.

- In our study, we have considered homogeneous servers for running the simulations. In future, We would like to extend our work to include heterogeneous runtime systems. Also, with the growing relevance of edge computing and digital twin techniques, efficient resource allocation and scheduling of the simulations at the edge will be necessary.

- In our study, we have also not taken into account any kind of fault and resilience techniques into account. Checkpointing mechanisms allow for saving the application state, which can be used to recover application state in case of failures. The EXPPO work can benefit from having such fault-tolerance techniques for recovering simulation states in case of failure.

CHAPTER VII

RESEARCH OUTREACH

VII.1 Introduction

VII.1.1 Complexities in Learning and Teaching Distributed Systems Algorithms

The design of large-scale networked and distributed systems must handle many complex issues, such as time synchronization, fault management, replication and replica synchronization, consensus among peers, concurrency control and race conditions, leader-election among nodes, deadlock avoidance, etc. Addressing these complex problems is further excarbated by the heterogeneity in distributed systems in terms of network topology (ring, star, mesh, etc.), node types (fixed vs mobile nodes, static vs dynamic nodes, physical vs virtual nodes), communication styles (client-server, peer-to-peer, publish-subscribe, etc.), and network types (Ethernet, WiFi, Satellite). The complexity of these design considerations and various accidental complexities make both the teaching and the learning of algorithms for distributed systems a daunting task for instructors and students, respectively.

From our experience both as students taking a course on Distributed Systems and as an instructor teaching such a course, we observed that existing teaching modalities, tools and techniques for understanding algorithms for distributed systems often rely on traditional approaches, such as didactic lecturing, simple proof sketches on the whiteboard, using theorem provers, and basic simulations or toy assignments in some programming language.

We believe that this approach incurs several difficulties for students including (1) often having to program the algorithms in programming languages they are not experts in, (2) analyzing these algorithms in simulators/emulators they are unfamiliar

with, and (3) needing to deal with accidental complexities in order to deploy them on real hardware to realistically validate them or propose improvements and extensions to them. Due to a piecemeal approach in learning and implementing these algorithms (i.e., programming/learning algorithms individually), students (1) cannot analyze multiple algorithms at the same time to compare and contrast them, (2) cannot seamlessly switch between simulation, emulation and real deployment on hardware, and (3) consequently do not obtain a holistic view of distributed systems and how different algorithms work together in real world distributed systems.

The instructor faces a different set of challenges. For instance, the inherent asynchrony and scale of distributed systems makes it hard for an instructor to show students the multiple different execution traces that a distributed system can illustrate in its life time. By no means do we imply that existing teaching modalities are not needed; rather what we propose is that instructors require some way in which they can keep the students engaged after which proof sketches and theorem provers can be utilized. We believe that hands-on, immersive learning [82] where students are allowed to conduct different kinds of "what-if" analyses (e.g., tweaking certain parameters of the distributed algorithm or tweaking some steps in an algorithm) and letting the students observe the impact of their slight modifications may provide a significantly more engaging and rewarding learning experience for the student, who may then feel obliged to utilize theorem provers and associated tools to prove the correctness of the original and/or the modified algorithms. Unfortunately, we have not come across any readily available capabilities that fulfill these critical needs for distributed systems learning and instruction.

VII.1.2 Solution Approach and Organization of Chapter

To address these challenges we need a learning and technology-based approach that will alleviate the need for students to learn an unfamiliar programming language

or a simulation tool to experiment with distributed systems. Secondly, such an approach should provide the student with intuitive and higher-levels of abstractions that are closer to the domain and semantics of distributed systems rather than having the student deal with low-level and mundane syntactic details of programming languages and simulation tools. Moreover, the approach should be extensible and enable the student to seamlessly move between simulation, emulation and real-world experimentation. Finally, the approach should promote maximal reuse so that students can build larger distributed systems by composing smaller but intuitive building blocks.

We surmise that these critical needs can be met using Software Product Lines (SPLs) [62] and model-driven engineering (MDE) [134] in the context of cloud platforms that will help improve teaching and learning of distributed systems algorithms. The key intuition behind applying SPL principles stems from the observation that these algorithms tend to share several common traits (e.g., communication paradigm, such as client-server versus publish/subscribe, or the model of computation, such as synchronous data flow or reactive asynchronous) while differing only in some aspects (e.g., the actual publish/subscribe technology used or the protocol encoding for messages).

Consequently, a collection of distributed systems algorithms can be viewed as variants of a product line. The challenge then lies in understanding and capturing the commonality and variability across these algorithms, and developing techniques needed to automate the synthesis of these variants so that the different dimensions of accidental complexities faced by the student can be substantially alleviated. MDE is used to realize these capabilities because it provides the user with intuitive, higher-level abstractions compared to programming languages to model the distributed systems algorithms and use generative techniques to almost completely automate the entire experimental setup including deployment and orchestration.

In the context of using MDE-based approaches, we have focused on visual modeling for imparting learning objectives to the students. Visual approaches for teaching have been found very effective in research projects, such as Betty's Brain [102], Code.org [96], NetsBlox [51, 52] and in our prior work on C3STEM [37, 73, 74, 138]. Both NetsBlox and C3STEM particularly make use of model-driven engineering techniques. To that end, our solution is a learning framework for distributed systems called the *Playground of Algorithms for Distributed Systems (PADS),* that reifies SPL principles by building on MDE with generative capabilities, feature modeling and teaching/learning tools and technologies.

In this work we complement and build upon our previous work [35] and make the following contributions:

- Concrete details on the design and implementation of the PADS framework beyond that provided in [35] including description of the underlying web-based modeling environment that provides new features such as web-based and collaborative modeling and the learning outcomes addressed by PADS (Section VII.3).
- Deeper insights into the runtime experimentation and deployment using the PADS framework (Section VII.4).
- Providing early results from a user study of the PADS framework in a classroom setting (Section VII.5).

VII.2 Related Work

In this section we compare PADS with related efforts along three key dimensions: existing work in teaching specific software for distributed systems algorithms, use of model driven engineering in the design of large-scale software systems in the context of educational learning systems, and visual learning aids.

Learning systems for distributed algorithms teaching: Authors in [66]

present a comprehensive survey providing an overview of different tools, simulators and learning platforms available for teaching distributed systems. It outlines tools available for managing deployment, execution, discovery, monitoring and configuration of distributed systems. It also presents a list of algorithms that can be used for teaching and demonstrating intricate details of distributed algorithms.

ViSiDiA [21] is a framework for designing, simulating and visualizing distributed algorithms. It is developed using Java language frameworks. It provides implementations of different distributed systems like sensor networks and mobile agents. A user can specify their custom distributed algorithms by making use of framework specific Java API. Distal [45] is another framework that is specifically aimed at a certain class within the distributed systems algorithm, namely fault-tolerant systems. It is developed on top of the Scala programming framework. One can write pseudo code for the algorithm using its DSML to translate into an executable code. The executable can then be deployed on clusters for testing. However, it lacks integration with simulators that would facilitate quick testing and debugging of algorithms.

Another teaching and learning framework called FADA (Framework Animations of Distributed Algorithms) is presented in [120]. In FADA, the simulations are written using Java programming language using the visualization APIs provided by the framework. It also provides a set of preassembled simulations for different algorithms which can be used as examples for demonstrating distributed algorithms to students.

The frameworks presented above have the following shortcomings compared to our approach. First, the distributed algorithms need to be written in a language which the framework supports. Secondly, the tools presented above do not support seamless translation of programming artifacts from simulation to real world deployment.

MDE in learning systems: Previous work has shown MDE, specifically, domain-specific modeling languages (DSMLs) have being effective tools [115] in developing teaching software systems. Students have also seen the benefits of rapid

code generation based on MDE techniques. In [80], students were able to rapidly synthesize code artifacts using MDE to rapidly generate code, when changes where required to be made in platform configuration of robotics control code and mobile device.

An educational game design software framework is presented in [95]. It utilizes a model driven approach to describe educational game concepts. It presents an educational game metamodel that defines platform-independent educational game concepts. The framework aims to design educational games to motivate the students to get involved in the learning process thereby effectively conveying educational material.

SPL techniques were applied for design, development and support of a family of elearning systems [59] called TALES. It also highlighted some of the challenges involved in the development of large-scale educational system and how SPL helped it to gain 10-fold productivity boost in the developmental efforts. The educational systems were built as a part of Adult Literacy Programme (ALP) for teaching illiterates in India in 22 Indian languages. Unlike the work presented above our area of study is focused on a special topic within computer science which is the distributed systems algorithms. Our work leverages the MDE and the SPL techniques in the design of a learning framework for distributed algorithms.

Visual Programming based learning frameworks: Computational thinking (CT) in the recent times, has become focus of many researchers including our prior work in the field of educational computing [36, 74, 131]. CT can be broadly summarized as a problem solving process involving abstraction of problem into smaller separation of concern entities, designing algorithm to solve the problem, simulating the solution approach and verification of the effectiveness of the solution to build a better solution strategy [153]. CT based framework such as the C3STEM [74] engages K-12 students to solve complex real world problem like the traffic flow modelling using a visual programming interface to design, simulate and analyze the solution strategy.

Yet another MDE/visual programming tool for learning is called NetsBlox [51, 52] which aims to provide a gentle introduction to distributed computing to K-12 students. PADS also uses the same MDE development environment as NetsBlox and in contrast to NetsBlox is focused on more rigorous distributed systems algorithms.

In [161], authors developed a visual programming toolkit called Alice, to teach object oriented programming to students. Their observation from the student subject study revealed that students learned new concepts in computer science effectively and also developed higher-order thinking skills using the tool. Another useful observation was that the drag-and-drop ability of designing new programs helped students from preventing making syntax errors in their program due to the auto-code generation. Another visual programming platform called code.org [96] has been very popular and serving millions of users worldwide in introducing basics of computer programming. The study in [96] also notes that students developed positive behavior towards programming and also improved their reflective thinking skills towards problem solving. Visual programming based teaching methodology has been found useful to expose students to new teaching concepts.

VII.3 Design and Implementation of PADS

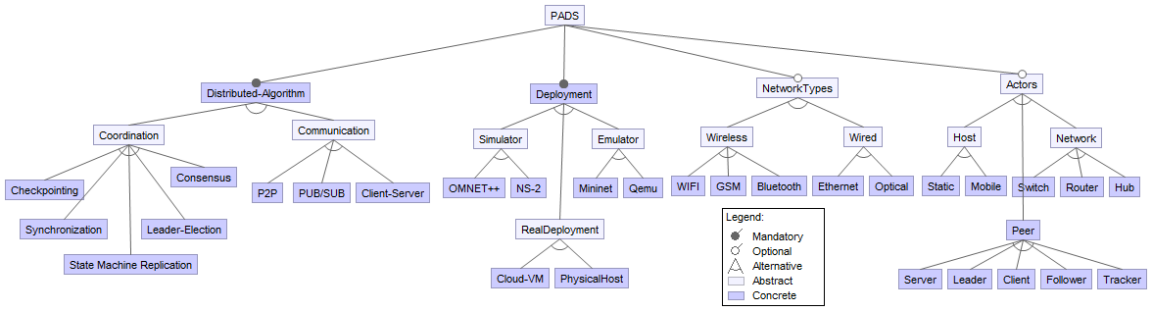

Figure 44: Feature Model Diagram of Playground of Algorithms for Distributed Systems (PADS) Framework

This section delves into the details of the design and implementation of the PADS framework. We first outline the key learning outcomes we intend to address via the framework. Next, we delve into the details of its model-based design including feature model representation and web based modeling environment. Lastly, we describe the roles and responsibilities of PADS's actors which includes students and instructors.

VII.3.1 Underlying Philosophy Behind PADS

In the educational teaching domain, learning objects have been an extremely useful resource in imparting education to the subjects [60, 72, 108, 141, 147]. In the literature, learning objects have been defined as "*Learning Objects (LOs) are digital resources that can be used (and reused) to support the learning process*" [65]. The learning objects have the property of re-usability, and sharing of learning material. As an instructional artifact, the learning objects should be able to easily incorporate in the design of larger educational teaching context. Generative learning objects (GLO) are a class of Learning Objects (LO) from which LO-specific functional implementation properties can be generated [65]. Learning objects are appealing in the space of learning computer science concepts.

Use of learning objects is still not that popular in the area of educational tools for computer science due to difficulties in designing learning objects as reusable software artifacts [141]. In a recent study [142], the authors have incorporated GLO in the teaching of robot programming. Authors have incorporated different robot control programs which are treated as LOs and the code generated for the robot target platform is treated as GLO. Model-driven engineering (MDE) has been shown to be an effective software engineering technique for reusing and sharing of software artifacts. MDE combined with code generation facility allows us to bring the learning paradigms of LO and GLOs in the practical use for our distributed algorithms learning toolkit. In the case of distributed algorithms teaching toolkit, different algorithms

models will form the LO, and their target execution environment specific program code will be GLO.

From the viewpoint of learning outcomes, we have focused on a subset of key learning outcomes outlines by the Accredition Board for Engineering and Technology (ABET) engineering criteria [63]. Specifically, for an effective instructional method for educating students, addressing the learning outcomes becomes key in achieving student learning objectives in an institutional course. PADS tries to address the following **General Criteria 3**, student learning outcomes from the ABET engineering program.

- *(a) an ability to apply knowledge of mathematics, science, and engineering.* PADS is designed to enable students to apply the concepts they learn in Distributed Systems and model the algorithm to learn its behavior.

- *(b) an ability to design and conduct experiments, as well as to analyze and interpret data.* PADS is design to enable a student to setup one or more experiments to evaluate a distributed systems algorithm scalably without incurring mundane and error-prone activities stemming from setting up one or more experiments.

- *(c) an ability to design a system, component, or process to meet desired needs.* PADS enables a student to tweak different parameters and modify existing algorithms to understand the impact on the resulting behaviors.

- *(k) an ability to use the techniques, skills, and modern engineering tools necessary for engineering practice.* PADS enables students to use modern tools including MDE tools, emulation environments such as Mininet, virtualization techniques such as virtual machines and containers, and study a variety of networking issues among others as they study distributed systems.

We now present the design and implementation of the Playground of Algorithms for Distributed Systems (PADS), which is an extensible framework that manages a

software product line of distributed algorithms used as an instructional and learning aid for distributed systems. It uses MDE and SPL techniques to integrate various distributed systems algorithms for teaching, and cloud platforms for deployment of experiments.

VII.3.2 Feature Model Representation

For a successful SPL for PADS, we need to manage the commonalities and variabilities that are exhibited for realizing the development, implementation and demonstration of distributed algorithms. One of the well-known approaches for representing and managing these commonalities and variabilities is by the means of feature models [40]. Feature models provide proven techniques for improving reusability by specifying reuse rules. The feature model for PADS must capture the commonalities and different dimensions of variabilities incurred in the learning process [35]. To that end we have defined a conceptual feature model for PADS as shown in Figure 44 and described in [35].

VII.3.3 Realizing the PADS Feature Model using Model-driven Engineering

The SPLs can be built using modular software. Changes in the feature configurations can be mapped to the changes in the software modules [150]. The design and development of modular software framework is a challenging task. We use model-driven engineering (MDE) techniques to codify the feature model by mapping it to metamodel(s) of a domain-specific modeling language (DSML) and use generative technologies, which are key artifacts of MDE, to automate the synthesis of product variants of our PADS product line. MDE helps to codify the necessary properties of problem domain which is decoupled from specific solution domain (e.g., simulator specific programming language, target hardware specific). MDE helps to integrate

domain knowledge into the metamodels and model interpreters [70]. The MDE design approach helps to map different configuration possibilities for a specific instance of SPL based on where it is deployed. This is useful when one needs to deploy a distributed algorithm software product family onto different platforms/simulators, MDE can bring all software components and their configuration mapped to the platform specific source codes and scripts.

Our PADS framework can be hosted on virtual machines or containers deployed on the Openstack cloud, which is an open source cloud computing infrastructure [146]. In our current prototype, we use containers. Cloud computing provides on-demand access to large pool of shared resources for compute intensive simulations and network experimentation. Students and instructors access these virtual machines/containers using remote access client which could be either a web browser client or a desktop client.

VII.3.4 Web Based Modeling Environment

In our previous work [56], we used the Generic Modeling Environment (GME) [101] as an environment to build and develop meta-models and models, and model interpreters. Despite the widespread use of GME in designing the DSML as evident by the works presented in [23, 24], it falls short in meeting our requirement of online and collaborative design and development of PADS artifacts. As such we decided to look into the next generation of GME being developed at Vanderbilt University called the WebGME [106].

WebGME is an online browser-based modeling environment. It supports features of GME like creating of meta-models, models, model visualization and model interpreters. Apart from this, it supports the collaborative modeling, version controlling of models, user management, cloud-based infrastructure, and model executors that can

run either on the client side web-browser or run in the resourceful cloud infrastructure depending on the user configuration.

WebGME[1] provides an environment to define the syntax and semantics of a DSML through metamodeling. Model interpreters can be defined associated with the meta-models that can provide additional semantics to the language which are not captured in a visual form as well as provide the generative capabilities needed for automation. The WebGME environment can be used to build model instances of a DSML. Thus, in our PADS framework, an instructor can extend existing metamodels for the collection of algorithms by providing a metamodel for a new algorithm. The students use the PADS framework to develop model instances and configure them for their experimental scenarios.

VII.3.5 Roles and Responsibilities of PADS's Actors

Figure 45 shows the different phases that comprises the use of the PADS framework for teaching distributed systems algorithms. An instructor who is a domain expert with a knowledge of model driven engineering concepts, lays down the foundation aspects as to which different feature model entities are involved in the design and deployment runtime of the algorithms. Based on the specific distributed algorithm concepts, using the PADS builtin meta-model library, the instructor creates a specific meta-model. Now, the software artifacts associated with the meta-model and the necessary model interpreter logic is also written by the instructor. Though we believe this would involve some learning curve for the instructor to get familiar with the PADS meta-modelling and model interpreter programming, the potential benefits of the PADS to the student learning outweighs this limitation. Moreover, we believe that a repository of such individual algorithm-specific metamodels can be envisioned and reused on a large scale.

[1]https://webgme.org/

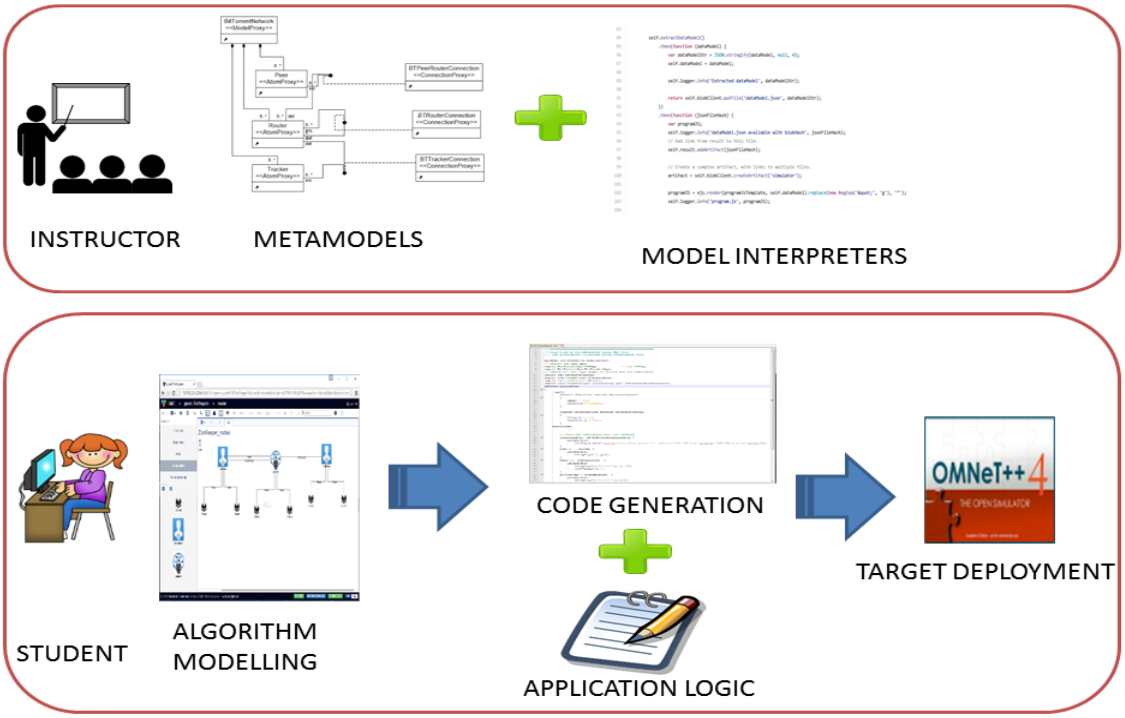

Figure 45: Model-based Process for Distributed Algorithm Demonstration and Deployment

Based on the type of the deployment (e.g., simulator specific), the associated deployment logic generators are also required to be a part of the model interpreters. Once, this initial design activity is complete, the tool is now ready to be used by the students. Students can now start modelling the distributed algorithm, such as different actors in the distributed systems, their attribute properties, characterizing the communication medium properties used during the simulation, network topology of the distributed systems, runtime platform for distributed systems deployment. Once the student has modelled the distributed systems algorithm, code generation facility can be utilized to generate the boilerplate code for the algorithm. The boilerplate code consists of the information specified during the modelling of the experiment and deployment runtime specific gluecode. Based on the learning activity, the instructor can then ask the students to write the application logic code for a functional

distributed systems algorithm using the generated boilercode. If the writing of the distributed algorithms is a too complex activity for the students' level of learning, the instructor may decide to provide the application logic code to the students. This phase helps the students in understanding the different programming constructs and the internals of the algorithm. Once the application code is ready for testing and deployment, the student can then invoke deployment specific set of model interpreters, which can then upload this application logic codes to the runtime execution platform for algorithm execution.

VII.4 Runtime Architecture of PADS

The PADS framework utilizes cloud computing infrastructure to deploy distributed systems experiments. The cloud computing provides elastic infrastructure facility which enables on-demand access to the computing resources. Figure 46 gives an overview of the layered architecture consisting of distributed systems experimentation, user interactions, cloud deployment and experimentations, and user analysis visualizations interface components of the PADS framework. Next we describe this layered architecture and the functionalities it provides to provide a scalable PADS framework.

VII.4.1 User Interaction Layer

The user interactions layer in the PADS framework provides following features: user/students/instructor access management and security, web front end to model and design the distributed algorithms, interaction and runtime visualization of the experiment deployed on the target simulator/emulator running on the cloud infrastructure, and performance monitoring logs of the completed experiment. User interactions primarily occur through the Internet enabled web browser. Next we will discuss each of the components of the user interaction layer.

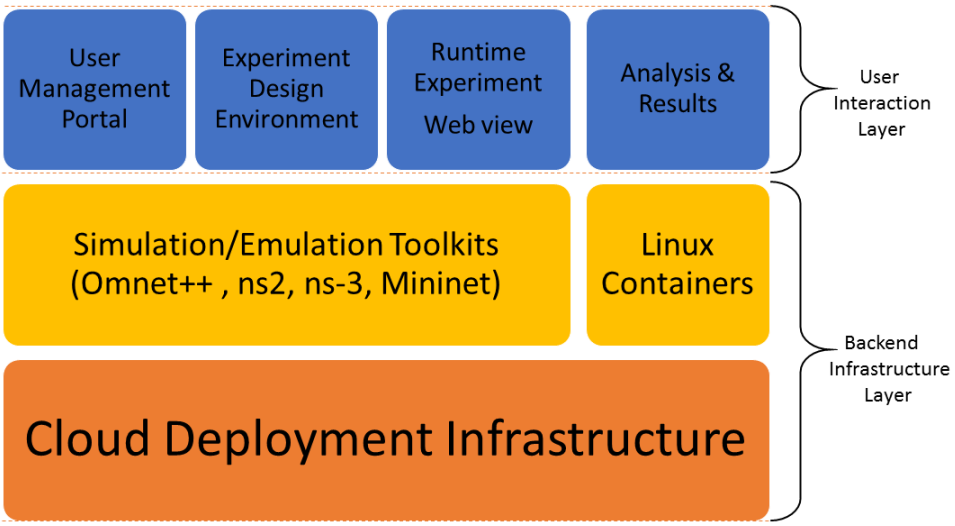

Figure 46: Cloud based architecture of PADS framework

1. User Management Portal: User management portal facilitates management of the user access privileges and login credentials to the PADS framework. User management portal lets an instructor manage login access for the students. Access rights to who can create, edit, and run experiments can be set as per the requirements of the distributed algorithm experiment for the students. If the instructor wants to give a 'read-only' access to the experiment scenario and prevent students from making any changes to the experiments, 'read-only' access can be set to the experiment project. Depending on the resource management and execution cost, the instructor may also want to set when and who can have access to the runtime infrastructure for deploying the distributed algorithms experiments. This gives the instructor a fine grain control over the PADS framework and various runtime components of the system.

2. Experiment Design Environment: This component provides a user with a web interface to design and orchestrate the distributed algorithm construction and specification. It provides various prebuilt artifacts to get quickly started with teaching and

learning of the distributed systems concepts and algorithms. Some of the predefined algorithms include Client-Server model, BitTorrent, Paxos, Zookeeper, Chord, Pub-Sub. It also provides distributed systems specific actors like Server, Client, Leader, Tracker, Router, Wired and wireless interfaces, hub, switch. Users can build their own distributed systems algorithms representation using the prebuilt artifacts. Apart from designing the distributed algorithm experiment, the user can also specify the target simulation or emulation platform the user would like to deploy the experiment on. The current prebuilt target support is included for Omnet++, Mininet, ns-2. Due to the MDE approach as discussed earlier, new target platform support can be easily extended.

3. Runtime Experiment Webviewer: The runtime experiment webviewer provides a web enabled view of the simulator/emulation runtime software. The distributed algorithm that is ready to be executed is deployed on the target platform in the cloud infrastructure. The user may need to see different runtime properties offered by the simulator. To enable remote access to the runtime view of the simulator in the web browser, we use Virtual Network Computing (VNC) technology. noVNC is an HTML5-compliant VNC client that enables remote desktop control and viewer via a web browser interface. Figure 47 shows the Runtime Experiment Webviewer running an Omnet++ based distributed algorithm experiment displayed in the web browser.

4. Analysis and Results: Analysis of the experiment can give various insights into performance and effectiveness of the distributed algorithm that is constructed. One can tweak different algorithm-specific parameters to meet the user objectives for the distributed algorithms (such as optimization, best fit). After the experiments finish executing, there is a need to have a good performance and result aggregation and presentation component which the user can use to identify potential benefits or bottlenecks of the target distributed systems algorithm. The Analysis and Results

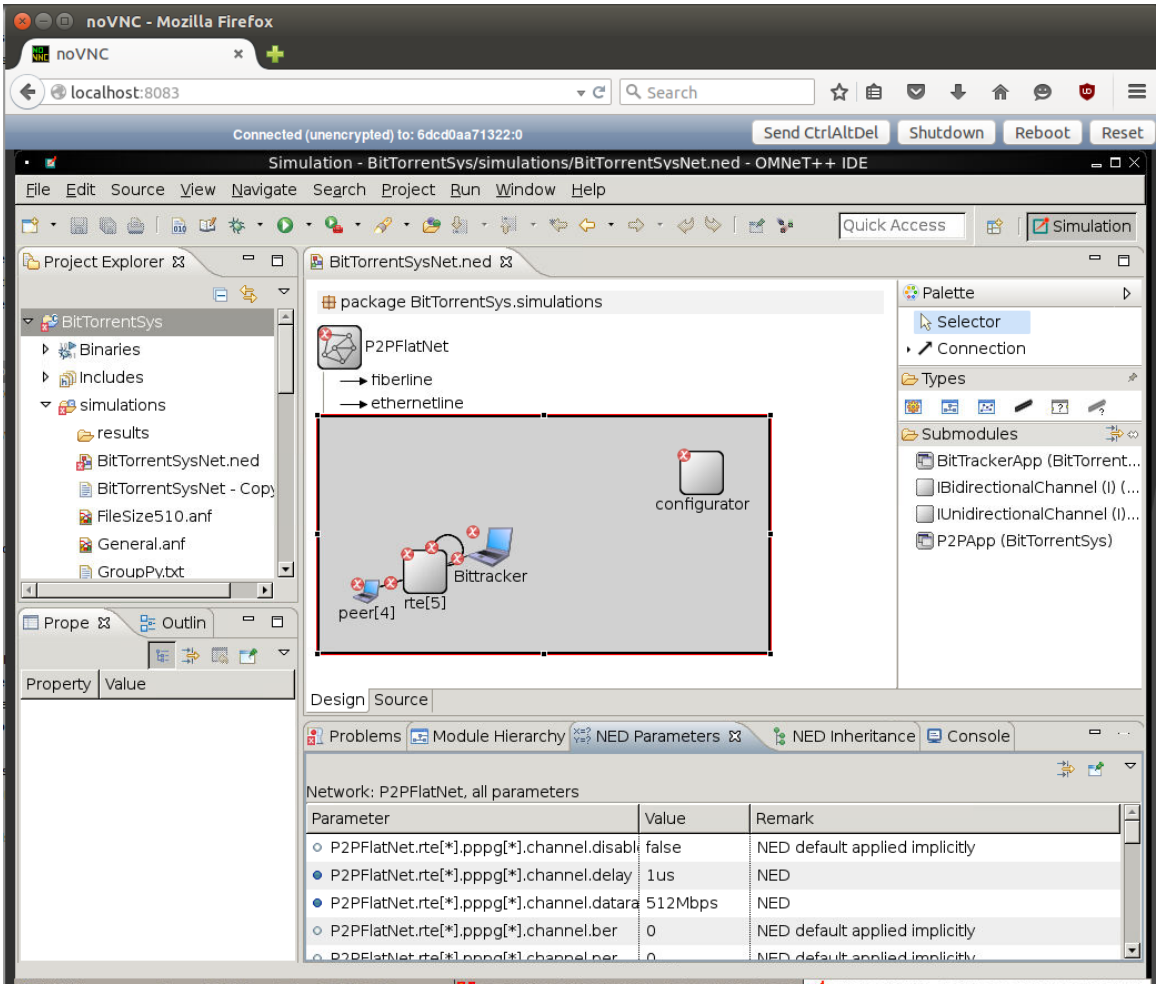

Figure 47: Runtime Experiment Webviewer illustrating Omnet++ simulator accessed through noVNC client

components collects and makes available various experimental metrics, results and logs generated by the target execution component.

VII.4.2 Backend Infrastructure Layer

This layer manages the deployment and runtime of the distributed algorithm experiments which are ready to be tested and run. The backend infrastructure deal mainly with providing elastic and extensible infrastructure to run the target simulators/emulators as configured by the user. To support running a large number of experiments which may consist of different simulation and emulation targets, one

needs a large number of computing resources to provide satisfiable Quality of User Experience (QoE). Cloud computing addresses these requirements and provides on-demand access to the pool of resources to carry out experiments. Moreover, the simulators/emulators that need to be run should be able to start instantaneously without considerably long startup times. We are currently using the Linux container technology to provide fast startup of simulators/emulators. Also, to manage a large pool or cluster of physical and virtual machines on which the experiments will be running, we will need resource management capability to deploy and manage this PADS infrastructure. Next we discuss each of the components of the backend infrastructure in detail.

1. Simulation/Emulation Target Toolkits: The experiments that the user orchestrates need to be executed on one of the simulators or emulators that it is designed for. The simulators are hosted in the cloud infrastructure. As such the users do not need to have any preinstalled applications (simulators/emulators) on their local development machines. Some of these targets need very high resources to execute. Running these target toolkits on the cloud relieves users from the need of owning resourceful machines to test and play with the distributed systems experiments. Simulator/emulators can be accessed through the Runtime Experiment Webviewer. Another benefit of running these tools in the cloud environment is the lower entry to barrier to play with the distributed systems algorithms as all the tools needed to run the experiment is already preinstalled and configured, and the user does not have to deal with complexities of the application installation process. OMNET++, Mininet, ns-2, ns-3 are some of the tools that are configured and ready for the users to experiment with.

2. Linux Containers: Linux containers provide a process-based virtualization that enables wrapping all the application dependencies in an isolated sandboxed environment. Linux containers provides many attractive features which are particularly

suited for the PADS framework simulator/emulation deployment in the cloud environment. We are using Docker implementation to leverage Linux container technology. Some of the features provided by the Docker technology includes lightweight deployment, instant bootup time, and sandboxed environment. The simulation/emulation targets are wrapped into a Docker container that can be deployed when the experiment is ready to be executed. Docker container also contains a running VNC server which is used by the runtime experiment webviewer to control and view the simulator/emulator execution. As shown in the figure we have an Omnet++ experiment running which is accessed via the runtime experiment webviewer powered by noVNC HTML5 client software.

3. Cloud Deployment Infrastructure: The PADS framework leverages cloud computing infrastructure to execute and run the distributed systems algorithm experiments and evaluation. The simulation/emulation targets which are wrapped into Docker containers become the deployment artifacts that can be executed on the cloud computing infrastructure. To enable efficient resource utilization of the available infrastructure (physical and virtual machines) we are using a cluster management engine called Apache Mesos. Mesos provides easy abstraction of available computing environments such as physical machine hosts, virtual machine hosts into a unified resource pool. It allows fine grained resource procurement and scheduling required for the execution of simulation/emulation runtime targets. Based on the requirement of the individual targets such as CPU, RAM, number of cores and storage, the target is deployed on the computing host that can provide these resources. Mesos provides REST application protocol interfaces (API) to enable simulation/emulation runtime to be executed, deployed and managed from the user interaction layer.

VII.5 Framework Validation

In this section we show using a preliminary user study the accrued benefits and effectiveness of PADS in terms of increasing user productivity, ease of use, usefulness in learning distributed systems algorithms, and users' interest in continuing to use modern tools for distributed systems algorithm study. In [35], we have also demonstrated how PADS can be extended to include a new algorithm and showed PADS' effectiveness in terms of the effort saved on the part of the instructor and learner in using the framework.

VII.5.1 Preliminary User Study

To understand the effectiveness of the PADS tool in teaching distributed systems algorithms, a preliminary user study was conducted as a part of the Distributed Systems Principles (CS-6381) graduate level course offered at Vanderbilt University, USA.[2] The study was conducted in the Spring 2017 semester. The class comprised graduate-level students in Computer Science. A total of 17 students participated in the study. In the study, students were asked to work on tasks which consisted of creating networking topologies for a publish/subscribe distributed systems scenario. The Publish/Subscribe paradigm is a key part in information dissemination in distributed systems.

VII.5.1.1 Task Assignment And Survey Questions

As part of the CS-6381 course, students are required to complete their assignments inside a Mininet network emulator and demonstrate scalable publish/subscribe using the ZeroMQ communication middleware. They are expected to showcase topic filtering, ownership strength of the publishers and history of topic samples in their assignment. Prior to our user study, students had completed this assignment wherein

[2] An exemption was approved by the Institutional Review Board for this study.

they were required to create application logic for a pub/sub system in Mininet environment. Thus, an assumption is made in this study that students have an understanding and hands-on experience in writing programs using the Mininet emulator and other associated technologies.

For the PADS formulation of this problem, we provide students with actors such as publisher, subscriber and broker. So the PADS framework contained these network actors in the distributed systems scenario to construct publish/subscribe distributed systems study. Prior to using PADS, for the assignment students were required to manually create Python language code to generate the network topology and the deployment logic to place the different publishers, subscribers and brokers on the different nodes of the topology. These steps must be repeated for every new topology under which they want to evaluate their algorithms, which are separately coded in Python per actor.

For our preliminary study, we have focused on relieving the student from the mundane, repetitive and error-prone tasks of writing code for generating the network topologies and deploying the actors. Instead, they are provided the PADS framework and asked to visually create the topologies while allowing the tool to generate the underlying code and deployment logic. Students are given two network topologies as shown in the Figures 48 and 49. An example of the code snippet generated from the PADS framework for one of the topologies is shown in Figure 50. Note that with more complex topologies, the logic for the topology generator becomes more complex and can be a cause of substantial effort spent in getting the topology right instead of spending time on learning and understanding the behavior of the algorithm being evaluated.

To test the usefulness of the PADS tool in improving user productivity, students were first asked to manually create the network topology generator program required by Mininet. As a second step, the students were asked to use the PADS tool to

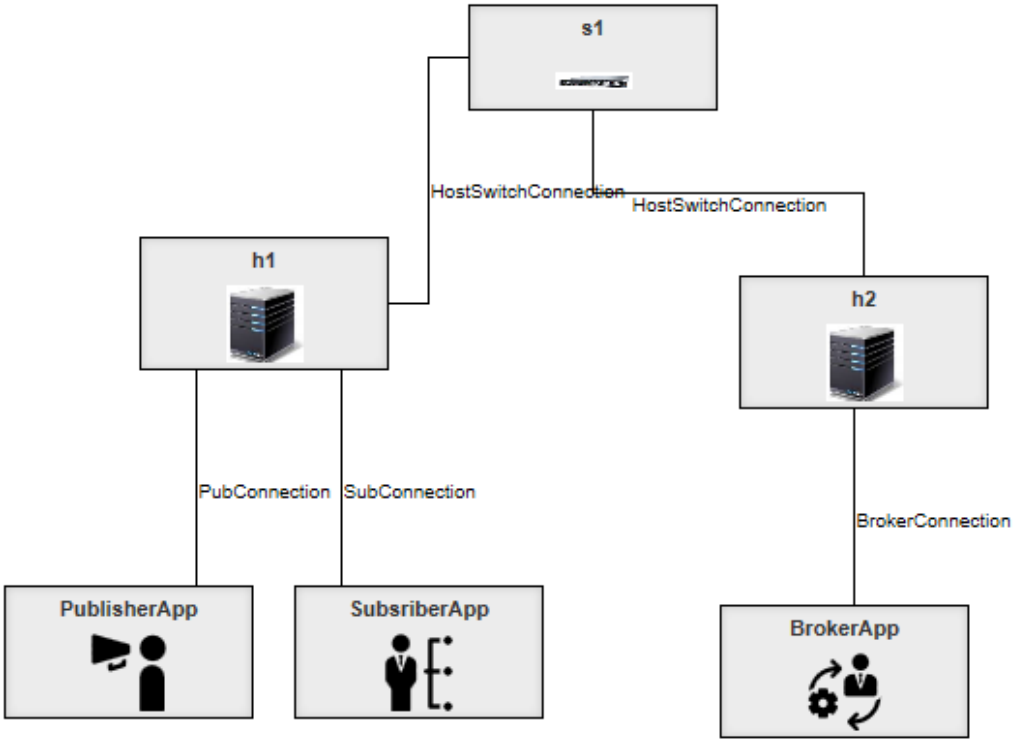

Figure 48: Network Topology One for user studies

create the same network topology. Students were asked to measure and report the approximate time required to complete the above tasks. At the end of the study, the users were asked to complete a survey form to report their experience in using the PADS framework.

The following questionnaire was created for the survey:

Q1: *Compare the time to complete tasks for Topology 1 and Topology 2 by completing Manually and then using PADS Framework.*

Q2: *Did PADS help you to avoid syntax errors in the topology generation process compared to writing of the topology file manually?* ***YES/NO***

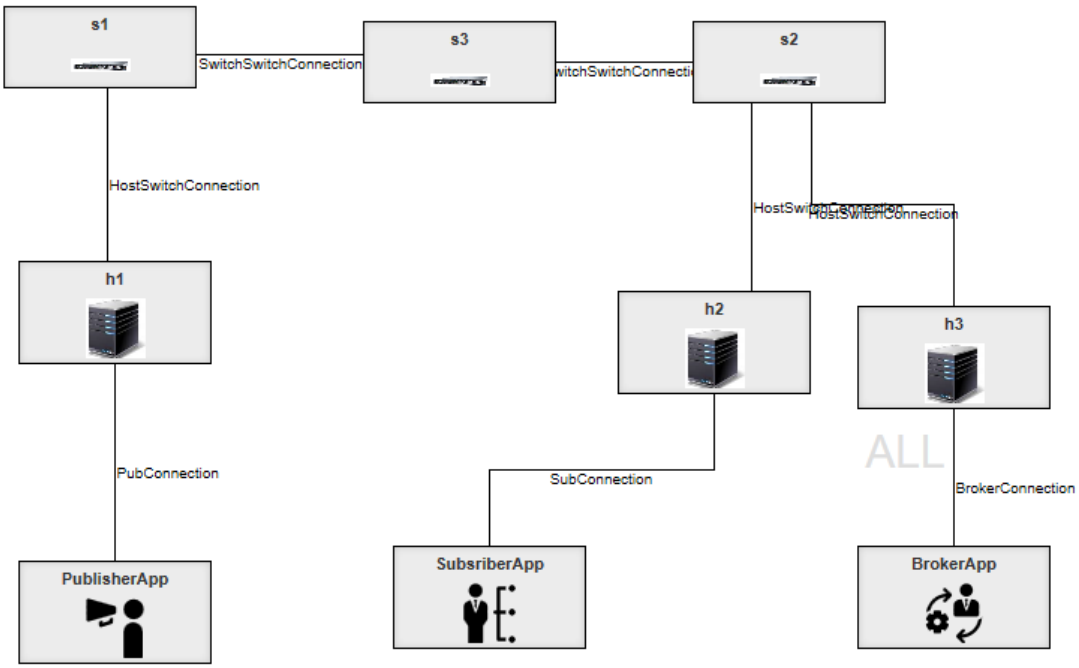

Figure 49: Network Topology Two for user studies

Q3: *How easy was it to use PADS?: scale **(1-10)**, where 1 is lowest, 10 is highest.*

Q4: *How likely are you to use PADS in future assignments/experimentation?: scale **(1-10)**, where 1 is lowest, 10 is highest.*

Q5: *Is PADS useful in learning distributed systems algorithms? **YES/NO***

The above questionnaire was carefully crafted to assess approximately how the PADS framework meets the ABET's learning outcomes as discussed in Section VII.3.1. **Q1** tries to address the learning outcome **3.k**, which assesses whether student is able to use the PADS system to create network topology experiments efficiently and effectively. **Q2** addresses the learning outcome **3.b**, as to how students where able to use the PADS framework to design the system correctly thereby avoiding manual mistakes during the design and setting up of the experiments. **Q3** and **Q4** assesses

the student's learning outcome **3.k**, as to how likely they are to use the PADS framework in future for solving distributed systems problems. **Q5** to tries to assess the learning outcomes **3.a** and **3.b**, which demonstrate students' eagerness to use PADS framework in creating distributed systems algorithms and finding solutions to the associated problems.

VII.5.1.2 Survey Results and Discussion

Based on the user study, the survey data was gathered and analyzed. Our findings are reported below.

Response to Q1→ Time to Complete: As can be seen in the box chart shown in Figure 51, comparison of time to completion for creating the topology files for the two test topologies using the manual approach and using the PADS tool is illustrated. The median value for time to completion using the manual approach is 7 and 9.5 minutes for the topologies 1 and 2, respectively. The 75th percentile value of the sample for time to completion using the manual approach resulted in 10.5 and 16.75 minutes for the topologies 1 and 2, respectively. Using the PADS approach, the median time required for completion for topologies 1 and 2 was 2 and 2 minutes, respectively. While the 75th percentile value of the sample for time to completion using the PADS approach is seen as 3.75 and 3.75, minutes respectively. Using these results, we can deduce that there is a substantial productivity gained using the PADS framework for constructing and creating the topology file for the distributed algorithms study.

Response to Q2→ Avoiding Manual Syntax Mistakes using PADS: As seen in Figure 52, 88 percent of the participants felt that the PADS helped them to avoid manual syntax mistakes in writing the topology description file. 12 percent of the participants did not answer the survey question. It can be seen that the PADS framework looks very appealing to the users in avoiding manual mistakes they might

make while writing the topology file manually. The PADS autogenerates the topology file and takes care of making sure the right parameters are passed to the connections creation function calls supported by the underlying target platform based on the topology specification described by the user.

Response to Q3→ Ease of use: As seen in Figure 53, most of the respondents gave a very high rating for the ease of use criteria of the PADS framework. 35.71 percent of users rated 7 and 8 on the scale of 10 for ease of use. While 21.43 percent and 7.14 percent rated it at 9 and 10 respectively on the scale of 10 for ease of use. This shows that our PADS framework is very easy to use for users to learn and play with distributed systems algorithms. The intuitive visual drag and drop environment helps the users to easily model different scenarios of the distributed systems for evaluation and study. Moreover, the support for the target emulator/simulator environment in PADS helps users to study distributed systems concepts in an environment which they are familiar with, which in this user study is Mininet.

Response to Q4→ Likely to continue using PADS tool: As seen in Figure 54, most of the respondents were excited to continue using the tool for future assignments and experimentations for learning distributed systems algorithms. 33.33 percent, 6.66 percent, 26.67 percent, 26.67 percent of the respondents rated 10, 9, 8 and 7 scores out of 10 rating respectively for likelihood in using PADS for future study purpose. While 6.67 percent of the participants rated 2 out of 10 score for using PADS in future for learning distributed systems algorithms. Overall, the trends show that students were able to experience the benefits offered by the PADS framework in the learning of the distributed systems algorithms. Due to this, the students developed great interest in working with the PADS tool for their future assignments and experiments.

Response to Q5→ Useful in Learning Distributed Systems Algorithms: As seen in Figure 55, 12 percent of the participants did not reply and 6 percent of

the participants were not certain to the question if the PADS tool will be useful in the learning of the distributed systems algorithms. While, 82 percent of the participants were of the opinion that PADS tool is useful in learning distributed systems algorithms. This strengthens our belief that the PADS framework designed for learning distributed systems algorithms will be a very resourceful tool for the learners of distributed systems algorithms. More studies on more complicated scenarios are necessary in future to corroborate our claims.

```python
#!/usr/bin/python

"""
This is a simple example that demonstrates multiple links
between nodes.
"""

from mininet.cli import CLI
from mininet.log import setLogLevel
from mininet.net import Mininet
from mininet.topo import Topo
from mininet.link import TCLink, TCIntf, Link


def runMultiLink():
    "Create and run multiple link network"
    topo = simpleMultiLinkTopo( n=2 )
    net = Mininet( topo=topo )
    net.start()
    CLI( net )
    net.stop()

class simpleMultiLinkTopo( Topo ):
    "Simple topology with multiple links"

    def __init__( self, n, **kwargs ):
        Topo.__init__( self, **kwargs )

        h1 = self.addHost('h1')
        h3 = self.addHost('h3')
        h2 = self.addHost('h2')

        s3 = self.addSwitch('s3')
        s2 = self.addSwitch('s2')
        s1 = self.addSwitch('s1')

        self.addLink(s1,s3,intf=TCIntf, params1 = { 'bw': 10 , 'delay' : '5ms' , 'loss' : 0}
        self.addLink(s3,s2,intf=TCIntf, params1 = { 'bw': 10 , 'delay' : '5ms' , 'loss' : 0}
        self.addLink(h2,s2,intf=TCIntf, params1 = { 'bw': 10 , 'delay' : '5ms' , 'loss' : 0}
        self.addLink(h3,s2,intf=TCIntf, params1 = { 'bw': 10 , 'delay' : '5ms' , 'loss' : 0}
        self.addLink(h1,s1,intf=TCIntf, params1 = { 'bw': 10 , 'delay' : '5ms' , 'loss' : 0}

if __name__ == '__main__':
    setLogLevel( 'info' )
    runMultiLink()
```

Figure 50: Source code generated for user study network topology

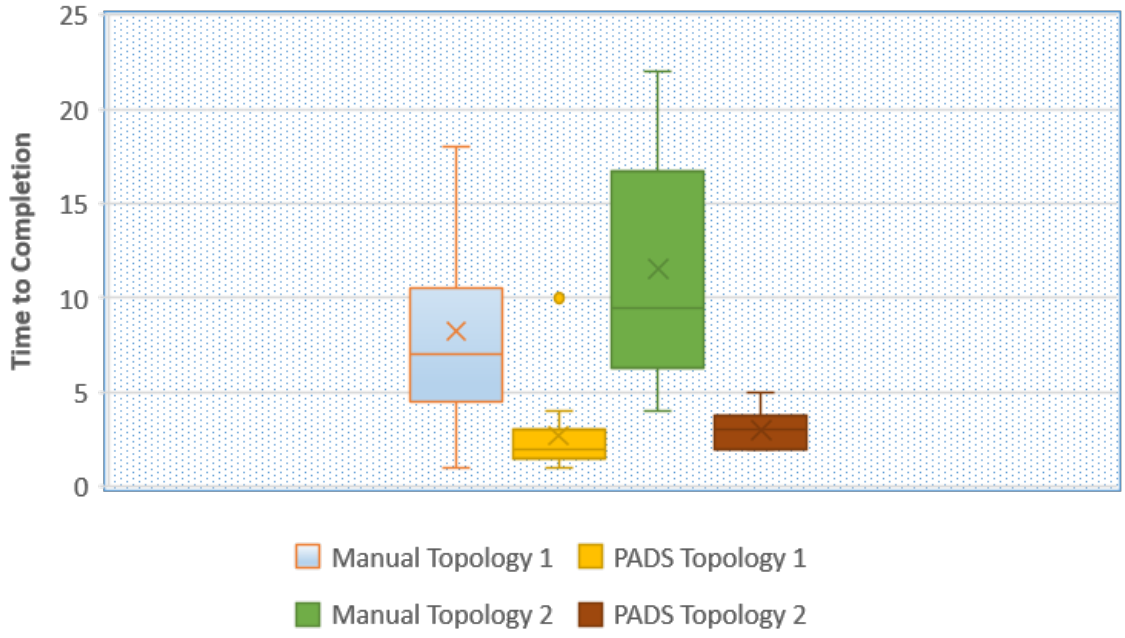

Figure 51: Survey response to Question 1: Compare time to complete tasks for Topology 1 and Topology 2 by completing Manually and then using PADS Framework

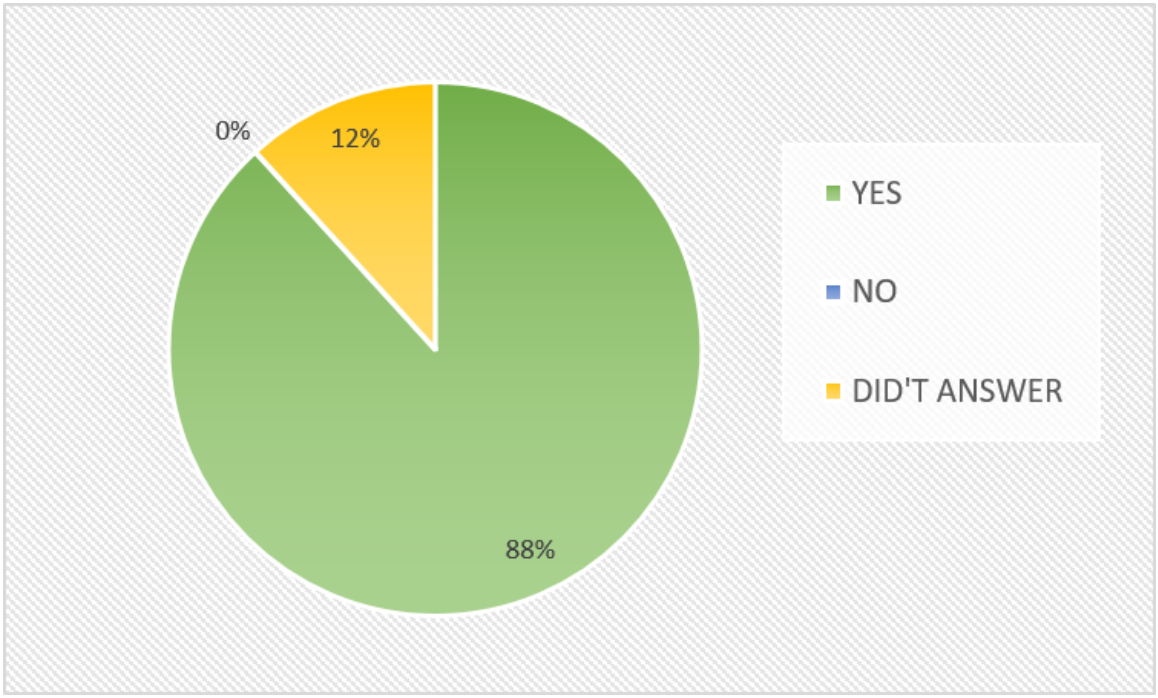

Figure 52: Survey response to Question 2: Did PADS help you to avoid manual syntax errors compared to writing of the topology file manually?

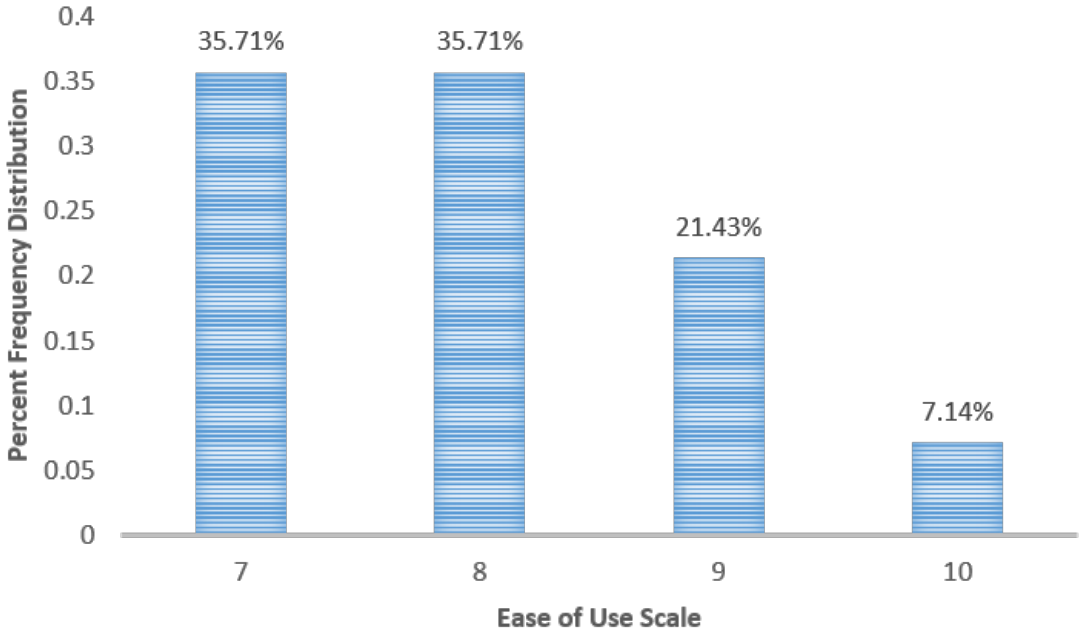

Figure 53: Survey response to Question 3: How easy was it to use PADS?

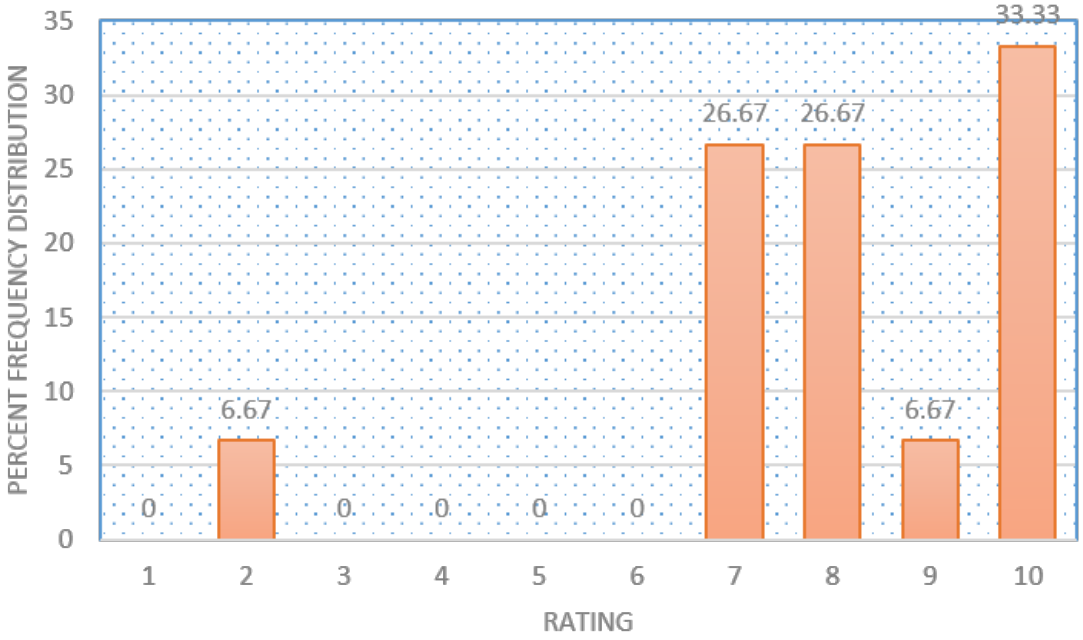

Figure 54: Survey response to Question 4: How likely are you to use PADS in future assignments/experimentation?

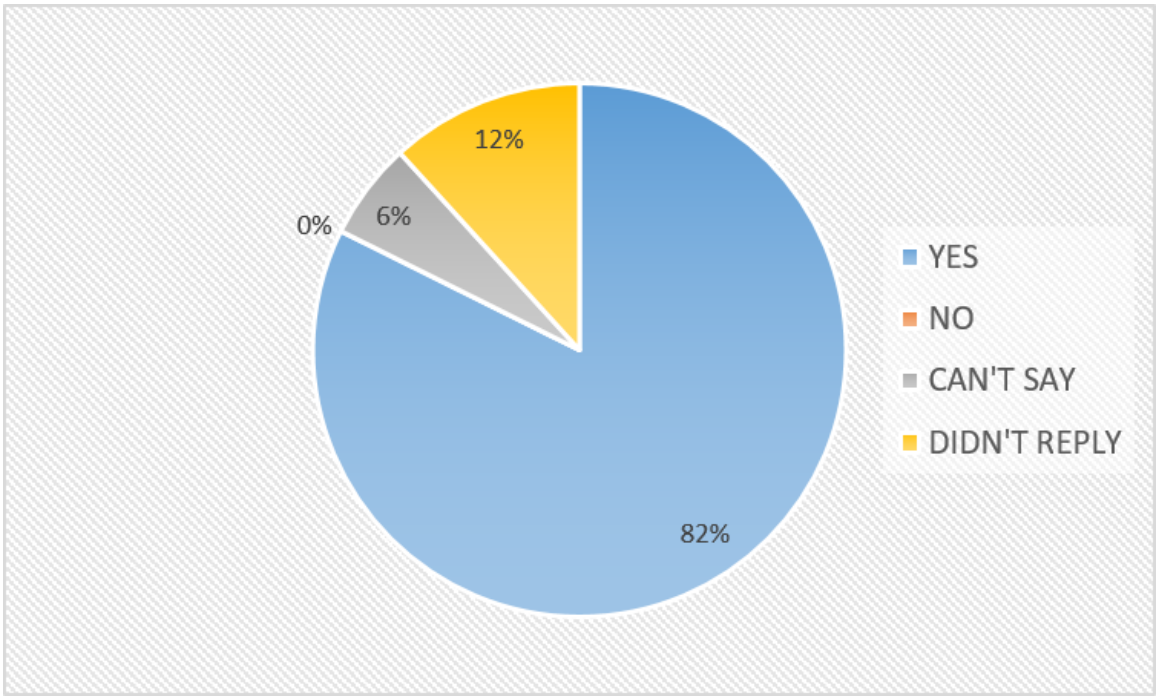

Figure 55: Survey response to Question 5: Is PADS useful in learning distributed systems algorithms?

VII.6 Concluding Remarks

This chapter motivated the need for an integrated teaching framework used for experimenting with distributed systems algorithms. To that end, this chapter describes the design, implementation and preliminary user evaluation of the *Playground of Algorithms for Distributed Systems (PADS)* framework, which provides intuitive, domain-specific modeling abstractions to capture various distributed systems algorithms, their components and requirements. Our preliminary user evaluation focused on understanding to what extent does PADS address the subset of learning outcomes we used from the ABET criteria. Our evaluation indicates that PADS prevents designers from making errors in the distributed systems algorithms test-bed setup and significantly simplifies system deployment by automating the generation of platform-specific metadata that faithfully implements the necessary execution dependency.

Ongoing and future work on PADS is focusing on framework extensibility so as to include more algorithms, reuse, building a repository of user models, conducting more user studies, and applying the framework beyond just distributed systems. PADS is available for download from:`https://github.com/doc-vu/pads`

CHAPTER VIII

CONCLUDING REMARKS

Many distributed applications in the cloud are faced with challenges that often are a combination of application-imposed, cloud environment-imposed and accessibility-related. In this doctoral research, we address a range of these challenges in the context of executing distributed applications in cloud computing platforms while ensuring that their quality of service requirements are met. In this chapter we summarize the key research contributions we have made in this doctoral dissertation.

VIII.1 Summary of Contributions

1. **FECBench:** To address the issues of application performance interference in multi-tenant cloud systems, we present a systematic methodology for benchmarking and building performance models for cloud hosted applications (See Chapter II). In this work we propose FECBench (Fog/Edge/Cloud Benchmarking), which allows users to build resource stressors that can stress multiple system resources all at once in a controlled manner. This allows to capture the effect of interference on application's performance. FECBench, using design of experiments (DoE) approach enables to users to build surrogate performance models thereby minimizing the profiling cost. Using emperical results we demonstrate the efficacy of FECbench in building performance model of applications.

2. **UPSARA:** To address the accessibility concerns in building performance models, we present the UPSARA framework (See Chapter III). UPSARA helps in

resolving accidental complexities involve in configuration, orchestration and deployment of monitoring instrumentation for scalable performance monitoring, analysis and testing framework for cloud-hosted applications. Using representative use cases we validated the effectiveness of the framework.

3. **PADS:** To address the accessibility concerns in rapid validation and testing for distributed systems, we present PADS platform (See Chapter IV). PADS leverages the principles of generative model-driven engineering and software product lines, and automates the synthesis of distributed systems simulations on cloud platforms. A prototype implementation of PADS is described to showcase using a use case, which shows the benefits of rapid deployment and automation in testing distributed systems.

4. **EXPPO and Co-simulation Design Studio:** To address the application and cloud imposed challenges in co-simulations, which are a type of distributed application, we present EXPPO (See Chapter VI). EXPPO allows for capturing performance traces of distributed co-simulations. Furthermore, EXPPO presents a technique for mitigating straggler scenarios in co-simulations by making appropriate resource allocation decisions. User's cost budget requirements are also taken into account for finding resource configuration for the co-simulations. We describe an optimization technique which finds the resource configuration for the co-simulation. Furthermore, EXPPO also supports a Co-simulation-as-a-Service middleware architecture. Using empirical studies, the effectiveness of EXPPO is demonstrated. We also address the accessibility concerns for design and modeling of co-simulation experiments by presenting a cloud-based design-studio, which allows for creating modeling and simulation of scenarios in co-simulations (See Chapter V).

5. **Research Outreach and Broader Impact:** We realized that the solutions

presented in PADS can be used as a teaching aid in classrooms for imparting education on distributed system concepts. Using an user study, we showed how PADS can be successfully leveraged to address complexities in teaching computer science concepts related to distributed systems (See Chapter VII).

VIII.2 List of Publications

Journal Publications

1. Barve, Yogesh D., Prithviraj Patil, Anirban Bhattacharjee, and Aniruddha Gokhale. "Pads: Design and implementation of a cloud-based, immersive learning environment for distributed systems algorithms." IEEE Transactions on Emerging Topics in Computing 6, no. 1 (2018): 20-31.

Conference Publications

1. Barve, Yogesh D., Prithviraj Patil, and Aniruddha Gokhale. "A cloud-based immersive learning environment for distributed systems algorithms." In Computer Software and Applications Conference (COMPSAC), 2016 IEEE 40th Annual, vol. 1, pp. 754-763. IEEE, 2016.

2. Barve, Yogesh, Shashank Shekhar, Shweta Khare, Anirban Bhattacharjee, and Aniruddha Gokhale. "UPSARA: A Model-Driven Approach for Performance Analysis of Cloud-Hosted Applications." In 2018 IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC), pp. 1-10. IEEE, 2018.

3. Barve, Yogesh, Shashank Shekhar,Ajay Chhokra, Shweta Khare, Anirban Bhattacharjee, Zhuangwei Kang, Hongyang Sun and Aniruddha Gokhale. "FECBench: A Holistic Interference-aware Approach for Application Performance Modeling." In 2019 IEEE 11th International Conference on on Cloud Engineering (IC2E),IEEE, 2019.

4. Patil, Prithviraj, Akram Hakiri, Yogesh Barve, and Aniruddha Gokhale. "Enabling software-defined networking for wireless mesh networks in smart environments." In 15th International Symposium on Network Computing and Applications (NCA 2016). 2016.

5. Shunxing Bao, Prasanna Parvathaneni, Yuankai Huo, Yogesh Barve, Andrew J. Plassard, Yuang Yao, Hongyang Sun, Ilwoo Lyu, David H. Zald, Bennett A. Landman and Aniruddha Gokhale. "Technology Enablers for Big Data, Multi-Stage Analysis in Medical Image Processing." Big Data (Big Data), 2018 IEEE International Conference

6. Khare, Shweta, Hongyang Sun, Julien Gascon-Samson, Kaiwen Zhang, Aniruddha Gokhale, Yogesh Barve, Anirban Bhattacharjee, and Xenofon Koutsoukos. "Linearize, predict and place: minimizing the makespan for edge-based stream processing of directed acyclic graphs." In Proceedings of the 4th ACM/IEEE Symposium on Edge Computing, pp. 1-14. 2019.

Short Papers, Workshop, Posters and Demonstrations

1. Anirban Bhattacharjee, Yogesh Barve, Shweta Khare, Shunxing Bao, Zhuangwei Kang, Aniruddha Gokhale, and Thomas Damiano, "STRATUM: A BigData-as-a-Service for Lifecycle Management of IoT Analytics Applications.", IEEE International Conference on BigData, Los Angeles, CA, USA,2019.

2. Aniruddha Gokhale, Yogesh Barve, Anirban Bhattacharjee, and Shweta Khare, "Software-defined and Programmable CPS/IoT OS: Architecting the Next Generation of CPS/IoT Operating Systems.", In 1st International Workshop on Next-Generation Operating Systems for Cyber-Physical Systems (NGOSCPS), Montreal, Canada, 2019.

3. Barve, Yogesh, Himanshu Neema, Stephen Rees, and Janos Sztipanovits. "Towards a Design Studio for Collaborative Modeling and Co-Simulations of Mixed Electrical Energy Systems." In 2018 IEEE International Science of Smart City Operations and Platforms Engineering in Partnership with Global City Teams Challenge (SCOPE-GCTC), pp. 24-29. IEEE, 2018.

4. Barve, Yogesh, Shashank Shekhar, Ajay Chhokra, Shweta Khare, Anirban Bhattacharjee, and Aniruddha Gokhale. "FECBench: An Extensible Framework for Pinpointing Sources of Performance Interference in the Cloud-Edge Resource Spectrum." In 2018 IEEE/ACM Symposium on Edge Computing (SEC), pp. 331-333. IEEE, 2018.

5. Bhattacharjee, Anirban, Yogesh Barve, Aniruddha Gokhale, and Takayuki Kuroda. "A Model-Driven Approach to Automate the Deployment and Management of Cloud Services." In 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion), pp. 109-114. IEEE, 2018.

6. Bhattacharjee, Anirban, Barve, Yogesh, Shweta Khare, Shunxing Bao, Aniruddha Gokhale, and Thomas Damiano. "Stratum: A Serverless Framework for the Lifecycle Management of Machine Learning-based Data Analytics Tasks." In 2019 USENIX Operational Machine Learning (OpML), USENIX, 2019.

7. Barve, Yogesh D., Himanshu Neema, Aniruddha S. Gokhale, and Janos Sztipanovits. "Model-driven Automated Deployment of Large-scale CPS Co-simulations in the Cloud." In Poster track of the ACM/IEEE 20th International Conference on Model-driven Engineering Languages and Systems (MODELS), Austin, TX, USA, Sept 1722, 2017, pp. 463464.

8. Barve, Yogesh D. "Towards Model Driven Verifiable Deployment of Distributed Simulations in Cloud.", In Doctoral Symposium track of the ACM/IEEE 20th

International Conference on Model-driven Engineering Languages and Systems (MODELS), Austin, TX, USA, Sept 1722, 2017, pp. 479.

9. Yogesh Barve, Shashank Shekhar, Ajay D. Chhokra, Shweta Khare, Anirban Bhattacharjee, and Aniruddha Gokhale, "Demo Paper: FECBench A Framework for Measuring and Analyzing Performance Interference Effects for Latency-Sensitive Applications", RTSS@Works Demo Session of the 39th IEEE Real-time Systems Symposium (RTSS), Nashville, TN, USA, Dec 1114, 2018, pp. 2.

10. Yogesh Barve, Himanshu Neema, Aniruddha Gokhale, and Janos Sztipanovits, "Towards an Automated Deployment Framework for Large-scale CPS Co-simulations in the Cloud", First Cyber Physical Systems Symposium (CyPhySS), Robert Bosch Center for Cyber Physical systems, Indian Institute of Science (IISc), Bengaluru, India, July 1921, 2017.

11. A. Bhattacharjee, Y. Barve, A. Gokhale, and T. Kuroda, "Cloudcamp: A model-driven generative approach for automating cloud application deployment and management," Technical Report, no. ISIS-17-105, Sept. 2017.

Tutorials and Talks

1. Yogesh Barve, Anirban Bhattacharjee, and Aniruddha Gokhale, PADS A Model Driven Engineering Framework for Learning Distributed Systems Algorithms, Tutorial at the ACM/IEEE 20th International Conference on Model-driven Engineering Languages and Systems (MODELS), Austin, TX, USA, Sept 1722, 2017, pp. 2.

2. Shashank Shekhar, Yogesh Barve, Shweta Khare, Anirban Bhattacharjee, and Aniruddha Gokhale, "FECBench: An Extensible Framework for Pinpointing Sources of Performance Interference in Cloud-to-Edge hosted Applications,"

Tutorial at IEEE International Conference on Cloud Engineering (IC2E), Orlando, FL, Apr 2018, pp. 1.

3. Shashank Shekhar, Yogesh Barve, and Aniruddha Gokhale, "Understanding Performance Interference Benchmarking and Application Profiling Techniques for Cloud-hosted Latency-Sensitive Applications", Tutorial at ACM/IEEE International Conference on Utility Cloud Computing (UCC), Austin, TX, Dec 2017, pp. 111120.

4. Yogesh Barve, Anirban Bhattacharjee, Shweta Khare, and Aniruddha Gokhale, "Investigating Dynamic Resource Management Solutions for Cloud Infrastructures using Chameleon Cloud," 2nd Chameleon Users Meeting, Austin, TX, USA, Feb 67, 2019, pp. 2.

5. Aniruddha Gokhale, Yogesh Barve, Anirban Bhattacharjee, and Travis Brummett, "Experiences using Chameleon in a Cloud Computing Course," 2nd Chameleon Users Meeting, Austin, TX, USA, Feb 67, 2019, pp. 2.

REFERENCES

[1] Stress-ng, 1999.

[2] IEEE Std 15162010, IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)- Framework and Rules, 2010.

[3] Ansible-automation for everyone. `https://www.ansible.com/`, 2018.

[4] Cache monitoring technology and cache allocation technology. `https://github.com/intel/intel-cmt-cat`, 2018.

[5] Collectd - the system statistics collection daemon, 2018.

[6] Cyber-Physical Systems Virtual Organization (CPS-VO), Feb. 2018.

[7] Docker, Feb. 2018.

[8] Embedded javascript templates. `http://ejs.co`, 2018.

[9] Grafana, Feb. 2018.

[10] Influxdb - time series database, 2018.

[11] JavaScript Object Notation (JSON), Feb. 2018.

[12] Jenkins, Feb. 2018.

[13] Keras - inceptionresnetv2, 2018.

[14] Keras application models. `https://keras.io/applications`, 2018.

[15] Likwid - system monitoring tool, 2018.

[16] OpenStack, Feb. 2018.

[17] Phoronix benchmark, 2018.

[18] Portico, Feb. 2018.

[19] Scikit-learn - machine learning library in python, 2018.

[20] IEEE Std 1516-2010. Ieee standard for modeling and simulation (m&s) high level architecture (hla)-framework and rules. Institute of Electrical and Electronic Engineers New York, 2010.

[21] W. Abdou, N.O. Abdallah, and M. Mosbah. Visidia: A java framework for designing, simulating, and visualizing distributed algorithms. In *Distributed Simulation and Real Time Applications (DS-RT), 2014 IEEE/ACM 18th International Symposium on*, pages 43–46, Oct 2014.

[22] Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring), pages 483–485, 1967.

[23] Kyoungho An and Aniruddha Gokhale. Model-driven performance analysis and deployment planning for real-time stream processing. In *Proceedings of the 19th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS13)*, pages 21–24, 2013.

[24] Kyoungho An, Takayuki Kuroda, Aniroddha Gokhale, Sumant Tambe, and Andrea Sorbini. Model-driven generative framework for automated omg dds performance testing in the cloud. *ACM Sigplan Notices*, 49(3):179–182, 2014.

[25] Varsha Apte, TVS Viswanath, Devidas Gawali, Akhilesh Kommireddy, and Anshul Gupta. Autoperf: Automated load testing and resource usage profiling of multi-tier internet applications. In *Proceedings of the 8th ACM/SPEC on*

International Conference on Performance Engineering, pages 115–126. ACM, 2017.

[26] Authors. Paper Title Anonymized for Double blind Reviewing. In *Appeared in ACM/IEEE Conference*, page 12, USA, October 2018.

[27] Wolfgang Barth. *Nagios: System and network monitoring*. No Starch Press, 2008.

[28] Yogesh Barve, Himanshu Neema, Stephen Rees, and Janos Sztipanovits. Towards a design studio for collaborative modeling and co-simulations of mixed electrical energy systems. In *2018 IEEE International Science of Smart City Operations and Platforms Engineering in Partnership with Global City Teams Challenge (SCOPE-GCTC)*, pages 24–29. IEEE, 2018.

[29] Yogesh Barve, Prithviraj Patil, Anirban Bhattacharjee, and Aniruddha Gokhale. Pads: Design and implementation of a cloud-based, immersive learning environment for distributed systems algorithms. *IEEE Transactions on Emerging Topics in Computing*, 2017.

[30] Yogesh Barve, Shashank Shekhar, Ajay Chhokra, Shweta Khare, Anirban Bhattacharjee, and Aniruddha Gokhale. Fecbench: An extensible framework for pinpointing sources of performance interference in the cloud-edge resource spectrum. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 331–333. IEEE, 2018.

[31] Yogesh Barve, Shashank Shekhar, Ajay Chhokra, Shweta Khare, Anirban Bhattacharjee, and Aniruddha Gokhale. Poster: Fecbench: An extensible framework for pinpointing sources of performance interference in the cloud-edge resource

spectrum. In *Proceedings of the Third ACM/IEEE Symposium on Edge Computing*. ACM, 2018.

[32] Yogesh Barve, Shashank Shekhar, Shweta Khare, Anirban Bhattacharjee, and Aniruddha Gokhale. Upsara: A model-driven approach for performance analysis of cloud-hosted applications. In *2018 IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC)*, pages 1–10. IEEE, 2018.

[33] Yogesh D. Barve, Himanshu Neema, Aniruddha Gokhale, and Sztipanovits Janos. Model-driven automated deployment of large-scale cps co-simulations in the cloud (poster). In *ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems*, Austin, TX, 09/2017 2017.

[34] Yogesh D Barve, Prithviraj Patil, Anirban Bhattacharjee, and Aniruddha Gokhale. Pads: Design and implementation of a cloud-based, immersive learning environment for distributed systems algorithms. *IEEE Transactions on Emerging Topics in Computing*, 6(1):20–31, 2018.

[35] Yogesh D Barve, Prithviraj Patil, and Aniruddha Gokhale. A cloud-based immersive learning environment for distributed systems algorithms. In *Computer Software and Applications Conference (COMPSAC), 2016 IEEE 40th Annual*, volume 1, pages 754–763. IEEE, 2016.

[36] Satabdi Basu, Gautam Biswas, Pratim Sengupta, Amanda Dickes, John S Kinnebrew, and Douglas Clark. Identifying middle school students challenges in computational thinking-based science learning. *Research and Practice in Technology Enhanced Learning*, 11(1):1–35, 2016.

[37] Satabdi Basu, Shashank Shekhar, Faruk Caglar, John Kinnebrew, Gautam

Biswas, and Aniruddha Gokhale. Collaborative Problem Solving Using a Cloud-based Infrastructure to Support High School STEM Education. In *ASEE Annual Conference K-12 and Pre-engineering Track*, pages 26.359.1–26.359.21, Seattle, WA, USA, June 2015. ASEE.

[38] Mihaly Berekmeri, Damián Serrano, Sara Bouchenak, Nicolas Marchand, and Bogdan Robu. Feedback autonomic provisioning for guaranteeing performance in mapreduce systems. *IEEE Transactions on Cloud Computing*, 2016.

[39] Benjamin Berg, Jan-Pieter L. Dorsman, and Mor Harchol-Balter. Towards optimality in parallel scheduling. *POMACS*, 1(2):40:1–40:30, 2017.

[40] Danilo Beuche, Holger Papajewski, and Wolfgang Schröder-Preikschat. Variability management with feature models. *Science of Computer Programming*, 53(3):333–352, 2004.

[41] A. Bhattacharjee, Y. Barve, A. Gokhale, and T. Kuroda. A model-driven approach to automate the deployment and management of cloud services. In *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, pages 109–114, Dec 2018.

[42] Anirban Bhattacharjee, Yogesh Barve, Aniruddha Gokhale, and Takayuki Kuroda. (wip) cloudcamp: Automating the deployment and management of cloud services. In *2018 IEEE International Conference on Services Computing (SCC)*, pages 237–240. IEEE, 2018.

[43] Anirban Bhattacharjee, Yogesh Barve, Shweta Khare, Shunxing Bao, Aniruddha Gokhale, and Thomas Damiano. Stratum: A serverless framework for the lifecycle management of machine learning-based data analytics tasks. In *2019*

USENIX Conference on Operational Machine Learning (OpML 19), pages 59–61, Santa Clara, CA, 2019. USENIX Association.

[44] Anirban Bhattacharjee, Yogesh D. Barve, Takayuki Kuroda, and Aniruddha Gokhale. Cloudcamp: A model-driven generative approach for automating cloud application deployment and management. Report, Institute for Software Integrated Systems, Vanderbilt University, Nashville, 2017.

[45] M. Biely, P. Delgado, Z. Milosevic, and A. Schiper. Distal: A framework for implementing fault-tolerant distributed algorithms. In *Dependable Systems and Networks (DSN), 2013 43rd Annual IEEE/IFIP International Conference on*, pages 1–8, June 2013.

[46] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. The parsec benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 72–81. ACM, 2008.

[47] Stephen M Blackburn, Robin Garner, Chris Hoffmann, Asjad M Khang, Kathryn S McKinley, Rotem Bentzur, Amer Diwan, Daniel Feinberg, Daniel Frampton, Samuel Z Guyer, et al. The dacapo benchmarks: Java benchmarking development and analysis. In *ACM Sigplan Notices*, volume 41, pages 169–190. ACM, 2006.

[48] Erik Blasch, Sai Ravela, and Alex Aved. *Handbook of dynamic data driven applications systems*. Springer, 2018.

[49] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16. ACM, 2012.

[50] Antonio Brogi and Stefano Forti. Qos-aware deployment of iot applications through the fog. *IEEE Internet of Things Journal*, 4(5):1185–1192, 2017.

[51] Brian Broll, Ákos Lédeczi, Péter Völgyesi, János Sallai, Miklós Maróti, Stephanie Wieden-Wright, Alexia Melo, and Chris Vanags. A Visual Programming Environment for Learning Distributed Programming. In *To Appear in the Proceedings of the 48th ACM Technical Symposium on Computing Science Education*. ACM, 2017.

[52] Brian Broll, Péter Völgyesi, János Sallai, and Akos Lédeczi. NetsBlox: a Visual Language and Web-based Environment for Teaching Distributed Programming. https://netsblox.org/NetsBloxWhitePaper.pdf, 2016.

[53] Tomasz Buchert, Cristian Ruiz, Lucas Nussbaum, and Olivier Richard. A survey of general-purpose experiment management tools for distributed systems. *Future Generation Computer Systems*, 45:1–12, 2015.

[54] Ngoc Bao Bui, Liming Zhu, Ian Gorton, and Yan Liu. Benchmark generation using domain specific modeling. In *Software Engineering Conference, 2007. ASWEC 2007. 18th Australian*, pages 169–180. IEEE, 2007.

[55] Frank Buschmann, Kelvin Henney, and Douglas Schimdt. *Pattern-oriented Software Architecture: on patterns and pattern language*, volume 5. John wiley & sons, 2007.

[56] Faruk Caglar, Kyoungho An, Shashank Shekhar, and Aniruddha Gokhale. Model-driven performance estimation, deployment, and resource management for cloud-hosted services. In *Proceedings of the 2013 ACM workshop on Domain-specific modeling*, pages 21–26. ACM, 2013.

[57] Faruk Caglar, Shashank Shekhar, and Aniruddha Gokhale. A performance interferenceaware virtual machine placement strategy for supporting soft realtime applications in the cloud.

[58] Marco Cavazzuti. Design of experiments. In *Optimization Methods*, pages 13–42. Springer, 2013.

[59] Sridhar Chimalakonda and Kesav V Nori. What makes it hard to apply software product lines to educational technologies? In *Product Line Approaches in Software Engineering (PLEASE), 2013 4th International Workshop on*, pages 17–20. IEEE, 2013.

[60] Thomas KF Chiu and Daniel Churchill. Design of learning objects for concept learning: Effects of multimedia learning principles and an instructional approach. *Interactive Learning Environments*, 24(6):1355–1370, 2016.

[61] Hyun Cho, Jeff Gray, and Eugene Syriani. Creating visual domain-specific modeling languages from end-user demonstration. In *Modeling in Software Engineering (MISE), 2012 ICSE Workshop on*, pages 22–28. IEEE, 2012.

[62] Paul Clements and Linda Northrop. Software product lines: practices and patterns. 2002.

[63] Engineering Accreditation Commission et al. Criteria for accrediting engineering programs. *Accreditation Board for Engineering and Technology Inc*, 1999.

[64] Drury B Crawley, Linda K Lawrie, Frederick C Winkelmann, Walter F Buhl, Y Joe Huang, Curtis O Pedersen, Richard K Strand, Richard J Liesen, Daniel E Fisher, Michael J Witte, et al. Energyplus: creating a new-generation building energy simulation program. *Energy and buildings*, 33(4):319–331, 2001.

[65] Robertas Damaševičius and Vytautas Štuikys. On the technological aspects of generative learning object development. In *International Conference on Informatics in Secondary Schools-Evolution and Perspectives*, pages 337–348. Springer, 2008.

[66] Francisco Javier De Hoyos Clavijo. Learning about distributed systems. 2010.

[67] Christina Delimitrou and Christos Kozyrakis. ibench: Quantifying interference for datacenter applications. In *2013 IEEE international symposium on workload characterization (IISWC)*, pages 23–33. IEEE, 2013.

[68] Christina Delimitrou and Christos Kozyrakis. Paragon: Qos-aware scheduling for heterogeneous datacenters. In *ACM SIGPLAN Notices*, volume 48, pages 77–88. ACM, 2013.

[69] Renan DelValle, Gourav Rattihalli, Angel Beltre, Madhusudhan Govindaraju, and Michael J Lewis. Exploring the design space for optimizations with apache aurora and mesos. In *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, pages 537–544. IEEE, 2016.

[70] Gan Deng, Douglas C. Schmidt, Aniruddha Gokhale, Jeff Gray, Yuehua Lin, and Gunther Lenz. Evolution in Model-Driven Software Product-line Architectures. In Pierre F. Tiako, editor, *Designing Software-Intensive Systems: Methods and Principles*. Idea Group, 2007.

[71] Docker. Docker swarm. `https://docs.docker.com/engine/swarm/`, 2020.

[72] Stephen Downes. Learning objects: resources for distance education worldwide. *The International Review of Research in Open and Distributed Learning*, 2(1), 2001.

[73] Anton Dukeman, Faruk Caglar, Shashank Shekhar, John Kinnebrew, Gautam Biswas, Doug Fisher, and Aniruddha Gokhale. Teaching Computational Thinking Skills in C3STEM with Traffic Simulation. In Andreas Holzinger and Gabriella Pasi, editors, *Human-Computer Interaction and Knowledge Discovery in Complex, Unstructured, Big Data*, volume 7947 of *Lecture Notes in Computer Science*, pages 350–357. Springer Berlin Heidelberg, 2013.

[74] Anton Dukeman, Liyan Hou, Shashank Shekhar, Faruk Caglar, John Kinnebrew, Gautam Biswas, Aniruddha Gokhale, and Doug Fisher. Modeling Student Program Evolution in STEM Disciplines. In *Poster paper at the 121st ASEE Annual Conference, K-12 and Pre-Engineering Track*. ASEE, Indianapolis, IN, USA, June 2014.

[75] Kaniz Fatema, Vincent C Emeakaroha, Philip D Healy, John P Morrison, and Theo Lynn. A survey of cloud monitoring tools: Taxonomy, capabilities and objectives. *Journal of Parallel and Distributed Computing*, 74(10):2918–2933, 2014.

[76] Michael Ferdman, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki, and Babak Falsafi. Clearing the clouds: a study of emerging scale-out workloads on modern hardware. In *ACM SIGPLAN Notices*, volume 47, pages 37–48. ACM, 2012.

[77] Vincenzo Ferme and Cesare Pautasso. A declarative approach for performance tests execution in continuous software development environments. In *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering*, pages 261–272. ACM, 2018.

[78] F Forster. Collectd open source project. `http://www.collectd.org`, 2017.

[79] Ian Foster, Carl Kesselman, Craig Lee, Bob Lindell, Klara Nahrstedt, and Alain Roy. A distributed resource management architecture that supports advance reservations and co-allocation. In *1999 Seventh International Workshop on Quality of Service. IWQoS'99.(Cat. No. 98EX354)*, pages 27–36. IEEE, 1999.

[80] Aniruddha S Gokhale and Jeff Gray. Advancing model driven development education via collaborative research. In *Educators' Symposium*, page 41. Citeseer, 2005.

[81] Sriram Govindan, Jie Liu, Aman Kansal, and Anand Sivasubramaniam. Cuanta: quantifying effects of shared on-chip resource interference for consolidated virtual machines. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, page 22. ACM, 2011.

[82] Brian Hand, Andy Cavagnetto, Ying-Chih Chen, and Soonhye Park. Moving past curricula and strategies: Language and the development of adaptive pedagogy for immersive learning environments. *Research in Science Education*, 46(2):223–241, 2016.

[83] Anca D Hansen, Clemens Jauch, Poul Sørensen, Florin Iov, and Frede Blaabjerg. Dynamic wind turbine models in power system simulation tool digsilent. *Report Risoe*, pages 1–80, 2003.

[84] Christoph Heger, André van Hoorn, Mario Mann, and Dušan Okanović. Application performance management: State of the art and challenges for the future. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*, pages 429–432. ACM, 2017.

[85] Graham Hemingway, Himanshu Neema, Harmon Nine, Janos Sztipanovits, and Gabor Karsai. Rapid synthesis of high-level architecture-based heterogeneous

simulation: a model-based integration approach. *Simulation*, 88(2):217–232, 2012.

[86] Mark D. Hill and Michael R. Marty. Amdahl's law in the multicore era. *Computer*, 41(7):33–38, 2008.

[87] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D Joseph, Randy H Katz, Scott Shenker, and Ion Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *NSDI*, volume 11, pages 22–22, 2011.

[88] Yang Hu, Huan Zhou, Cees de Laat, and Zhiming Zhao. Concurrent container scheduling on heterogeneous clusters with multi-resource constraints. *Future Generation Computer Systems*, 102:562–573, 2020.

[89] Alexandru Iosup, Radu Prodan, and Dick Epema. Iaas cloud benchmarking: approaches, challenges, and experience. In *Cloud Computing for Data-Intensive Applications*, pages 83–104. Springer, 2014.

[90] Alexandru Iosup, Nezih Yigitbasi, and Dick Epema. On the performance variability of production cloud services. In *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*, pages 104–113. IEEE, 2011.

[91] Seyyed Ahmad Javadi and Anshul Gandhi. Dial: Reducing tail latencies for cloud applications via dynamic interference-aware load balancing. In *Autonomic Computing (ICAC), 2017 IEEE International Conference on*, pages 135–144. IEEE, 2017.

[92] Deepal Jayasinghe, Galen Swint, Simon Malkowski, Jack Li, Qingyang Wang, Junhee Park, and Calton Pu. Expertus: A generator approach to automate

performance testing in iaas clouds. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 115–122. IEEE, 2012.

[93] Hiranya Jayathilaka, Chandra Krintz, and Richard M Wolski. Detecting performance anomalies in cloud platform applications. *IEEE Transactions on Cloud Computing*, (1):1–1, 2018.

[94] Albert Jonathan, Mathew Ryden, Kwangsung Oh, Abhishek Chandra, and Jon Weissman. Nebula: Distributed edge cloud for data intensive computing. *IEEE Transactions on Parallel and Distributed Systems*, 28(11):3229–3242, 2017.

[95] Mladjan Jovanovic, Dusan Starcevic, Miroslav Minovic, and Velimir Stavljanin. Motivation and multimodal interaction in model-driven educational game design. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 41(4):817–824, 2011.

[96] Filiz Kalelioğlu. A new way of teaching programming skills to k-12 students. *Comput. Hum. Behav.*, 52(C):200–210, November 2015.

[97] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian E Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica B Hamrick, Jason Grout, Sylvain Corlay, et al. Jupyter notebooks-a publishing format for reproducible computational workflows. In *ELPUB*, pages 87–90, 2016.

[98] Boris Koldehofe, Marina Papatriantafilou, and Philippas Tsigas. Lydian: An extensible educational animation environment for distributed algorithms. *J. Educ. Resour. Comput.*, 6(2), June 2006.

[99] Xenofon KoutsouKos, Gabor Karsai, Aron Laszka, Himanshu Neema, Bradley Potteiger, Peter Volgyesi, Yevgeniy Vorobeychik, and Janos Sztipanovits. SURE: A modeling and simulation integration platform for evaluation of secure

and resilient cyber–physical systems. *Proceedings of the IEEE*, 106(1):93–112, 2018.

[100] Kubernetes. Kubernetes :production-grade container orchestration. `https://kubernetes.io/`, 2020.

[101] Akos Ledeczi, Miklos Maroti, Arpad Bakay, Gabor Karsai, Jason Garrett, Charles Thomason, Greg Nordstrom, Jonathan Sprinkle, and Peter Volgyesi. The generic modeling environment. In *Workshop on Intelligent Signal Processing, Budapest, Hungary*, volume 17, 2001.

[102] Krittaya Leelawong and Gautam Biswas. Designing learning by teaching agents: The betty's brain system. *International Journal of Artificial Intelligence in Education*, 18(3):181–208, 2008.

[103] Xiangyu Lin and Chase Qishi Wu. On scientific workflow scheduling in clouds under budget constraint. In *2013 42nd International Conference on Parallel Processing*, pages 90–99. IEEE, 2013.

[104] Wei-Liem Loh et al. On latin hypercube sampling. *The annals of statistics*, 24(5):2058–2080, 1996.

[105] Jonathan Mace and Rodrigo Fonseca. Universal context propagation for distributed system instrumentation. In *Proceedings of the Thirteenth EuroSys Conference*, page 8. ACM, 2018.

[106] Miklós Maróti, Róbert Kereskényi, Tamás Kecskés, Péter Völgyesi, and Akos Lédeczi. Online collaborative environment for designing complex computational systems. *Procedia Computer Science*, 29:2432–2441, 2014.

[107] Jason Mars, Lingjia Tang, Robert Hundt, Kevin Skadron, and Mary Lou Soffa.

Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations. In *Proceedings of the 44th annual IEEE/ACM International Symposium on Microarchitecture*, pages 248–259. ACM, 2011.

[108] Rory McGreal. *Online education using learning objects*. Psychology Press, 2004.

[109] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2, 2014.

[110] Nicolas Michael, Nitin Ramannavar, Yixiao Shen, Sheetal Patil, and Jan-Lung Sung. Cloudperf: A performance test framework for distributed and dynamic multi-tenant environments. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*, pages 189–200. ACM, 2017.

[111] Asit K Mishra, Joseph L Hellerstein, Walfredo Cirne, and Chita R Das. Towards characterizing cloud backend workloads: insights from google compute clusters. *ACM SIGMETRICS Performance Evaluation Review*, 37(4):34–41, 2010.

[112] Nikita Mishra, John D Lafferty, and Henry Hoffmann. Esp: A machine learning approach to predicting application interference. In *Autonomic Computing (ICAC), 2017 IEEE International Conference on*, pages 125–134. IEEE, 2017.

[113] Nathaniel Morris, Christopher Stewart, Lydia Chen, Robert Birke, and Jaimie Kelley. Model-driven computational sprinting. In *Proceedings of the Thirteenth EuroSys Conference*, page 38. ACM, 2018.

[114] Y. Moses, Z. Polunsky, A. Tal, and L. Ulitsky. Algorithm visualization for distributed environments. In *Information Visualization, 1998. Proceedings. IEEE Symposium on*, pages 71–78, 154, Oct 1998.

[115] Sébastien Mosser, Philippe Collet, and Mireille Blay-Fornarino. Exploiting the

internet of things to teach domain-specific languages and modeling.

[116] Hiroyuki Nagataki, Taichi Fujii, Yukiko Yamauchi, Hirotsugu Kakugawa, and Toshimitsu Masuzawa. A kinesthetic-based collaborative learning system for distributed algorithms. In *Education Technology and Computer (ICETC), 2010 2nd International Conference on*, volume 2, pages V2–97. IEEE, 2010.

[117] Himanshu Neema, Gabor Karsai, and Alexander H Levis. Next-generation command and control wind tunnel for courses of action simulation. Technical Report no. ISIS-15-119, Institute for Software-Integrated Systems, Vanderbilt University, 2015.

[118] Himanshu Neema, Janos Sztipanovits, Martin Burns, and Edward Griffor. C2WT-TE: A model-based open platform for integrated simulations of transactive smart grids. In *Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES), 2016 Workshop on*, pages 1–6. IEEE, 2016.

[119] Dejan Novakovic, Nedeljko Vasic, Stanko Novakovic, Dejan Kostic, and Ricardo Bianchini. Deepdive: Transparently identifying and managing performance interference in virtualized environments. In *Proceedings of the 2013 USENIX Annual Technical Conference*, number EPFL-CONF-185984, 2013.

[120] Fionnuala O'Donnell. Simulation frameworks for the teaching and learning of distributed algorithms. 2006.

[121] Oleksii Oleksenko, Dmitrii Kuvaiskii, Pramod Bhatotia, and Christof Fetzer. Fex: A software systems evaluator. In *Dependable Systems and Networks (DSN), 2017 47th Annual IEEE/IFIP International Conference on*, pages 543–550. IEEE, 2017.

[122] Rihards Olups. *Zabbix Network Monitoring*. Packt Publishing Ltd, 2016.

[123] Opentracing. Opentracing specification. `https://opentracing.io/specification/`, 2020.

[124] Johan Pouwelse, Paweł Garbacki, Dick Epema, and Henk Sips. The bittorrent p2p file-sharing system: Measurements and analysis. In *Peer-to-Peer Systems IV*, pages 205–216. Springer, 2005.

[125] Xing Pu, Ling Liu, Yiduo Mei, Sankaran Sivathanu, Younggyun Koh, Calton Pu, and Yuanda Cao. Who is your neighbor: Net i/o performance interference in virtualized clouds. *IEEE Transactions on Services Computing*, 6(3):314–329, 2013.

[126] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[127] Thierry Rakotoarivelo, Guillaume Jourjon, and Max Ott. Designing and orchestrating reproducible experiments on federated networking testbeds. *Computer Networks*, 63:173–187, 2014.

[128] Thierry Rakotoarivelo, Maximilian Ott, Guillaume Jourjon, and Ivan Seskar. Omf: a control and management framework for networking testbeds. *ACM SIGOPS Operating Systems Review*, 43(4):54–59, 2010.

[129] Gourav Rattihalli. Exploring potential for resource request right-sizing via estimation and container migration in apache mesos. In *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, pages 59–64. IEEE, 2018.

[130] Kasim Rehman, Orthodoxos Kipouridis, Stamatis Karnouskos, Oliver Frendo, Helge Dickel, Jonas Lipps, and Nemrude Verzano. A cloud-based development

environment using hla and kubernetes for the co-simulation of a corporate electric vehicle fleet. In *2019 IEEE/SICE International Symposium on System Integration (SII)*, pages 47–54. IEEE, 2019.

[131] Alexander Repenning, David Webb, and Andri Ioannidou. Scalable game design and the development of a checklist for getting computational thinking into public schools. In *Proceedings of the 41st ACM technical symposium on Computer science education*, pages 265–269. ACM, 2010.

[132] Robert Ricci, Gary Wong, Leigh Stoller, Kirk Webb, Jonathon Duerig, Keith Downie, and Mike Hibler. Apt: A platform for repeatable research in computer science. *SIGOPS Oper. Syst. Rev.*, 49(1):100–107, January 2015.

[133] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

[134] Douglas C Schmidt. Model-driven engineering. *COMPUTER-IEEE COMPUTER SOCIETY-*, 39(2):25, 2006.

[135] S Selvarani and G Sudha Sadhasivam. Improved cost-based algorithm for task scheduling in cloud computing. In *2010 IEEE International Conference on Computational Intelligence and Computing Research*, pages 1–5. IEEE, 2010.

[136] Shashank Shekhar, Yogesh Barve, and Aniruddha Gokhale. Understanding performance interference benchmarking and application profiling techniques for cloud-hosted latency-sensitive applications. In *Proceedings of the10th International Conference on Utility and Cloud Computing*, pages 187–188. ACM, 2017.

[137] Shashank Shekhar, Yogesh Barve, and Aniruddha Gokhale. Understanding performance interference benchmarking and application profiling techniques for

cloud-hosted latency-sensitive applications. In *Proceedings of the10th International Conference on Utility and Cloud Computing*, UCC '17, pages 187–188, New York, NY, USA, 2017. ACM.

[138] Shashank Shekhar, Faruk Caglar, Anton Dukeman, Liyan Hou, Aniruddha Gokhale, John Kinnebrew, Gautam Biswas, and Doug Fisher. An Evaluation of a Collaborative STEM Education Framework for High and Middle School Students. In *Poster Paper at 121st ASEE Annual Conference, K-12 and Pre-Engineering Track*. ASEE, Indianapolis, IN, USA, June 2014.

[139] Marcio Silva, Michael R Hines, Diego Gallo, Qi Liu, Kyung Dong Ryu, and Dilma Da Silva. Cloudbench: Experiment automation for cloud environments. In *Cloud Engineering (IC2E), 2013 IEEE International Conference on*, pages 302–311. IEEE, 2013.

[140] Milan Sonka, Vaclav Hlavac, and Roger Boyle. *Image processing, analysis, and machine vision*. Cengage Learning, 2014.

[141] Vytautas Štuikys. *Smart Learning Objects for Smart Education in Computer Science: Theory, Methodology and Robot-Based Implementation*. Springer, 2015.

[142] Vytautas Štuikys, Renata Burbaitė, Kristina Bespalova, and Giedrius Ziberkas. Model-driven processes and tools to design robot-based generative learning objects for computer science education. *Science of Computer Programming*, 2016.

[143] Lavanya Subramanian, Vivek Seshadri, Arnab Ghosh, Samira Khan, and Onur Mutlu. The application slowdown model: Quantifying and controlling the impact of inter-application interference at shared caches and main memory. In *Proceedings of the 48th International Symposium on Microarchitecture*, pages

62–75. ACM, 2015.

[144] Yu Sun, Jules White, Sean Eade, and Douglas C Schmidt. Roar: A qos-oriented modeling framework for automated cloud resource allocation and optimization. *Journal of Systems and Software*, 116:146–161, 2016.

[145] Simon JE Taylor, Azam Khan, Katherine L Morse, Andreas Tolk, Levent Yilmaz, Justyna Zander, and Pieter J Mosterman. Grand challenges for modeling and simulation: simulation everywherefrom cyberinfrastructure to clouds to citizens. *Simulation*, 91(7):648–665, 2015.

[146] The OpenStack Project. Openstack: The open source cloud operating system.

[147] Rubén Peredo Valderrama, Leandro Balladares Ocaña, and Leonid B Sheremetov. Development of intelligent reusable learning objects for web-based education systems. *Expert Systems with Applications*, 28(2):273–283, 2005.

[148] Leslie G Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.

[149] Jürgen Walter, Simon Eismann, Johannes Grohmann, Dušan Okanovic, and Samuel Kounev. Tools for declarative performance engineering. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, pages 53–56. ACM, 2018.

[150] Jules White, James H Hill, Jeff Gray, Sumant Tambe, Aniruddha S Gokhale, and Douglas C Schmidt. Improving domain-specific language reuse with software product line techniques. *Software, IEEE*, 26(4):47–53, 2009.

[151] Johannes Wienke, Dennis Wigand, Norman Koster, and Sebastian Wrede. Model-based performance testing for robotics software components. In *2018*

Second IEEE International Conference on Robotic Computing (IRC), pages 25–32. IEEE, 2018.

[152] Tiffani L Williams and Rebecca J Parsons. The heterogeneous bulk synchronous parallel model. In *International Parallel and Distributed Processing Symposium*, pages 102–108. Springer, 2000.

[153] Jeannette M Wing. Computational thinking. *Communications of the ACM*, 49(3):33–35, 2006.

[154] Yair Wiseman and Dror G. Feitelson. Paired gang scheduling. *IEEE transactions on parallel and distributed systems*, 14(6):581–592, 2003.

[155] Miguel G Xavier, Kassiano J Matteussi, Fabian Lorenzo, and Cesar AF De Rose. Understanding performance interference in multi-tenant cloud databases and web applications. In *Big Data (Big Data), 2016 IEEE International Conference on*, pages 2847–2852. IEEE, 2016.

[156] Ran Xu, Subrata Mitra, Jason Rahman, Peter Bai, Bowen Zhou, Greg Bronevetsky, and Saurabh Bagchi. Pythia: Improving datacenter utilization via precise contention prediction for multiple co-located workloads. In *Proceedings of the 19th International Middleware Conference*, pages 146–160. ACM, 2018.

[157] Neeraja J Yadwadkar, Bharath Hariharan, Joseph E Gonzalez, Burton Smith, and Randy H Katz. Selecting the best vm across multiple public clouds: A data-driven performance modeling approach. In *Proceedings of the 2017 Symposium on Cloud Computing*, pages 452–465. ACM, 2017.

[158] Hailong Yang, Alex Breslow, Jason Mars, and Lingjia Tang. Bubble-flux: Precise online qos management for increased utilization in warehouse scale computers. In *ACM SIGARCH Computer Architecture News*, volume 41, pages

607–618. ACM, 2013.

[159] Amir Yazdanbakhsh, Divya Mahajan, Hadi Esmaeilzadeh, and Pejman Lotfi-Kamran. Axbench: A multiplatform benchmark suite for approximate computing. *IEEE Design & Test*, 34(2):60–68, 2017.

[160] Saad Zaheer, Asad Waqar Malik, Anis Ur Rahman, and Safdar Abbas Khan. Locality-aware process placement for parallel and distributed simulation in cloud data centers. *The Journal of Supercomputing*, 75(11):7723–7745, 2019.

[161] Jacky Xi Zhang, Li Liu, Patricia Ordonez de Pablos, and Jinghuai She. The auxiliary role of information technology in teaching: Enhancing programming course using alice. *International Journal of Engineering Education*, 30(3):560–565, 2014.

[162] Jiacheng Zhao, Huimin Cui, Jingling Xue, Xiaobing Feng, Youliang Yan, and Wensen Yang. An empirical model for predicting cross-core performance interference on multicore processors. In *Proceedings of the 22nd international conference on Parallel architectures and compilation techniques*, pages 201–212. IEEE Press, 2013.