AN APPROACH FOR AUTOMATING THE CALIBRATION OF SIMULATIONS OF PARALLEL AND DISTRIBUTED COMPUTING SYSTEMS

A THESIS SUBMITTED TO THE GRADUATE DIVISION OF THE
UNIVERSITY OF HAWAIʻI AT MĀNOA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

COMPUTER SCIENCE

MAY 2021

By

William Koch

Thesis Committee:

Henri Casanova, Chairperson
Edo Biagioni, Peter Sadowski

Keywords: PDC, Simulation, Automated Calibration, Computer Science

ACKNOWLEDGMENTS

ABSTRACT

Modern science is heavily intertwined with the use of computing, with complex and large-scale computational applications needing to be executed on parallel and distributed computing platforms. The execution of these applications on these platforms is facilitated by multi-component software infrastructures that implement various algorithms for orchestrating and managing application computation and data access. Given the complexity of such systems (i.e., application workload, compute platform, and software infrastructure), optimizing for their efficient execution is a difficult proposition, which raises many research questions. A large literature focuses on answering these questions, following an experimental approach: draw conclusions based on real-world experiments, that is, executions on real-world platforms. Real-world experiments have many shortcomings, including high cost, labor, and time. Furthermore, they cannot be used to explore hypothetical scenarios that go beyond application, platform, and workflow configurations at hand.

One approach that obviates the shortcomings of real-world experiments is simulation, i.e., the use of a software artifact that models and mimics the functional and performance behaviors of the execution of a parallel and distributed computing system. The main concern, however, is that of the accuracy of the simulation. High simulation accuracy can only be achieved by "calibrating" simulation parameters adequately with respect to real-world executions. Unfortunately, simulation calibration is rarely done in the literature, or, when it is done, it is a poorly documented, painstaking, manual process. In this thesis we explore the feasibility of automated simulation calibration in the context of the simulation of parallel and distributed computing systems. We frame the simulation calibration problem as an optimization problem, and propose an automated simulation calibration approach that can be instantiated for arbitrary simulation accuracy metrics and calibration algorithms. We evaluate our proposed approach via a case study for a particular production setting, namely the execution of scientific workflow applications via a workflow management system on a cluster managed by a batch scheduler. We find that our proposed approach is able to compute an accurate calibration for any given scenario, but we also find that simulation accuracy is diminished when using the computed calibration for simulating other scenarios (i.e., different application workloads, different platform scales). We investigate the reasons for this behavior, which point to fertile ground for future research.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

1.1 Context and Motivation

Almost every field of modern scientific research has become heavily intertwined with computing in order to collect and analyze data efficiently. The scale of such data processing can be immense, and as a result countless Parallel and Distributed Computing (PDC) scientific applications have been developed. One particularly prevalent and popular class of such applications is scientific workflows or *workflows*. A workflow is a series of compute tasks that are connected via data and control dependencies, such that there is an order, or flow, in which tasks need to execute. Workflows have been used in a wide range of fields (physics, biology, chemistry, seismology, etc), and are relied upon by thousands of researchers [26] for managing analyses, simulations, and other computations in almost every scientific domain [61, 43, 3, 68]. Workflows have underpinned some of the most significant discoveries of the last decade, including the first detection of gravitational waves from colliding black holes [24], the discovery of the Higgs boson [39], and the detection of an exotic nuclear decay [21]. The success of workflows in enabling new discoveries has fed into the confluence of demands society has for ever more powerful computing hardware and infrastructure.

As workflows have been adopted by a number of scientific communities, they have become more complex. For example, workflows increasingly integrate machine learning models to guide analysis, couple simulation and data analysis codes, and exploit specialized computing hardware (e.g., GPUs, neuromorphic chips, etc.) [28, 74]. Some workflows can now analyze terabyte-scale data sets, be composed of millions of individual tasks, require coordination between heterogeneous tasks, manage tasks that execute for milliseconds to hours, and can process data streams, files, and data placed in object stores. Workflow tasks can be single-core programs, loosely coupled parallel tasks (like MapReduce), or tightly coupled parallel tasks (as in MPI-based parallel programs) all within a single workflow, and can run on a range of platform configurations. As a result, workflows require sophisticated execution management capabilities. These capabilities are provided by a workflow management system (WMS), i.e., a software tool to orchestrate the workflow execution on compute resources. The importance of such systems cannot be overstated, and they have required decades of research, development, and community engagement to reach a high level of refinement [23, 43, 2, 6, 11, 45, 22, 72, 7].

Due to their computational demands, many workflows need to be executed on High-Performance Computing (HPC) platforms. The most commonly used such platforms are *clusters*: an aggregation of possibly large numbers of individually modest commodity computers, all connected together via a fast network. Clusters allow researchers to utilize the full benefits of parallel processing for executing large-scale workflows. However, these clusters are typically shared among many users and stakeholders, which requires careful management. The exact nature of how each cluster is managed varies, although there are some general considerations that any system must take into account. Those considerations, and the administration of the platform as a whole, are the responsibility of a particular system often termed Resource and Job Management Software (RJMS). While a WMS deals with the execution of individual tasks within a specific workflow, a RJMS imposes higher level management of workloads from different users. A prime example of such RJMS are "batch schedulers", a well-known example of which is the Simple Linux Utility for Resource Management (Slurm) [71]. Often, clusters are used for High-Throughput (HTC) applications. These are applications that consist of very larger number of tasks, where each task can execute only on a single compute node in a cluster. This is by contrast with applications in which "tasks" are in fact parallel applications themselves, i.e., can execute concurrently on multiple compute nodes. Many workflows are HTC applications. An example of resource management software for HTC is HTCondor [62]. Like Slurm, it can handle job submissions from competing users, and implements a variety of job prioritization, scheduling, and accounting policies.

Given the complexity of workflow applications and of the platforms on which they need execute, executing these workflows efficiently on these platforms is challenging. This challenge requires answering many algorithmic and/or systems research questions, e.g., when designing and implementing WMSs. Furthermore, users and their institutions may want to optimize for a specific type of workflow and/or provision a specific cluster architecture, which also raises difficult questions. Answering these questions requires performing experiments in which one executes actual workflows on actual platforms so as to draw various lessons and conclusions. Unfortunately, performing these experiments comes with many shortcomings. First, these experiments can be extremely expensive (e.g., monetary cost, power consumption) and time-consuming. Second, they can also be labor-intensive as they require precise (re)configurations of both applications and platforms. Third, and perhaps most importantly, one needs to perform experiments for hypothetical scenarios, so as to explore a wide range of settings that go beyond workflow and platforms at hand. This is so that generalizable conclusions can be drawn from the experiments, as well as conclusions that can hold for emerging and future application and platform settings.

An alternative approach to real-world experiments is *simulation*, i.e., the use of a software artifact that models and mimics the functional and performance behaviors of software and hardware stacks of interest. In our case, we consider the simulation of the execution of a workflow application on an HPC platform. Simulation obviates the shortcomings of real-world experiments, and in particular makes it possible to explore completely arbitrary experimental scenarios. The main concern with simulation, however, is that of its *accuracy*. If simulation results are not sufficiently accurate, then they are not useful, or even harmful as they would lead to false conclusions that would not hold in practice.

In the field of PDC, many simulation frameworks are available for developing simulators of applications executions on various platforms [64, 34, 73, 5, 10, 12, 52, 14, 51, 38, 47, 56, 48, 8, 16, 15, 17]. Some of these frameworks have striven to implement simulation models that achieve high accuracy. For instance, SimGrid [16, 58] has been actively developed for over 20 years and provides foundational simulation models and abstractions for simulating distributed applications, systems, and platforms. A main focus of SimGrid has been simulation accuracy, and thus it has been the object of a large number of (in)validation studies [9, 67, 66, 31, 40, 55, 27, 57, 33, 60, 59]. As a result, the simulation models in SimGrid should be usable to implement accurate simulations of, for instance, workflow executions on HPC platforms.

The driving motivation for this thesis is that although accurate simulation models are available, it is still challenging to use them to produce an accurate simulator. This is because each simulation model, such as the ones implemented in SimGrid, comes with several configuration parameters. These parameters govern the behavior of the model, and must be chosen carefully so as to capture the behavior of the target system of interest. Regardless of the potentially high accuracy of a particular simulation model, if values for its configuration parameters are not picked judiciously, accuracy will be poor. We term the process by which one picks these values *simulation calibration*. Calibration is a crucial part of simulation, and yet, to the best of our knowledge, at least in the field of PDC, it has received very little attention. Simulation calibration is often seen as a “dark art” and simulation calibration procedures are rarely documented in the literature. One of the reasons is that simulation calibration, when done manually, is extremely labor-intensive.

Thesis statement

In this thesis we develop a method for automated calibration of PDC simulators. This method consists of picking judicious simulation model parameter values using ground-truth real-world experimental data. Although general, we develop and evaluate our approach in the specific context of workflow applications executed on cluster platforms using WMS, HTC, and RJMS systems.

1.2 Roadmap

The remainder of this thesis is organized as follows:

- Chapter 2 discusses related work. In particular it reviews the state of the art of simulation of PDC systems, and then discusses current simulation calibration approaches used for PDC research (or lack thereof);

- Chapter 3 gives our problem statement and then describes our proposed automated simulation calibration approach. This approach can be instantiated with various definitions of simulation accuracy and with various simulation calibration algorithms. The chapter describes a few options for both, which are used for evaluating our proposed approach experimentally.

- Chapter 4 presents a case study with a real-world system. The broad objective of this case study is to evaluate the feasibility and effectiveness of our proposed approach when applied to complex production systems. Specifically, this chapter outlines key research questions, and then presents experimental results pertinent to each question.
- Chapter 5 summarizes our contributions and highlights directions for future work.

CHAPTER 2
RELATED WORK

There is a large literature devoted to the simulation of PDC systems. The main focus of the published work is on proposing simulation models and simulation frameworks, and on evaluating their usability, scalability and/or accuracy. By comparison, simulation calibration, while a necessary part of simulation-driven research and the primary focus on this thesis, is rarely discussed in the literature. We first review general previous work on PDC simulation (Section 2.1) and then previous work on simulation calibration (Section 2.2).

2.1 PDC Simulation Research

Many PDC simulation frameworks have been been developed to enable the simulation of specific classes of PDC applications [10, 12, 52, 14, 50, 51, 44, 38, 47, 56, 18, 15, 17]. These frameworks implement simulation models and abstractions, and provide APIs that facilitate the development of simulators. These simulators are then used for studying the functional and performance behaviors of application workloads executed on various hardware/software infrastructures.

The two main concerns for simulation are *accuracy* (the ability to faithfully reproduce real-world executions) and *scalability* (the ability to simulate large/long real-world executions quickly and with low RAM footprint). The simulation frameworks cited above achieve different compromises between the two. At one extreme are discrete-event models that capture "microscopic" behaviors of hardware/software systems (e.g., packet-level network simulation, block-level disk simulation, cycle-accurate CPU simulation), which favor accuracy over scalability. At the other extreme are analytical models that capture "macroscopic" behaviors via mathematical models. While these latter models lead to fast simulation, they must be developed carefully if high levels of accuracy are to be achieved [67].

The work in this thesis is agnostic to the simulation models and frameworks, since calibration must be performed regardless. However, to drive the development and demonstrate the feasibility of our proposed simulation approach, we use the SimGrid and WRENCH simulation frameworks. The years of research and development invested in the popular SimGrid simulation framework [18], have culminated in a set of state-of-the-art macroscopic simulation models that yield high accuracy, as demonstrated by (in)validation studies and comparisons to competing frameworks [9, 67, 66, 31, 41, 55, 27, 57, 33, 60]. Therefore, high simulation accuracy should be within reach for SimGrid simulators provided that simulation calibration is done effectively. The WRENCH simulation framework builds on SimGrid to provide high-level APIs that make it possible to implement simulators of complex PDC system with low effort. Since we evaluate our proposed approach via a case study for a complex PDC system, we use WRENCH to make the development of our simulator tractable.

2.2 Simulation Calibration

Regardless of the simulation frameworks used and the models they implement, simulators come with many parameters that must be ”calibrated” in the hope of achieving high simulation accuracy. The calibration problem can be defined as follows. Consider a real-world execution trace of an application workflow on a compute infrastructure (both hardware and software), and a simulator of that same execution that produces a simulated execution trace. This simulator's behavior is configured via a number of simulation parameters. There is a measure of the discrepancy, or “error”, between the real-world and the simulated execution traces. The objective is to find the parameter values that minimize this error. In other terms, the input to the simulation calibration problem is one real-world execution trace, and its output is a set of simulation parameter values.

The lower a simulation model's level of abstraction, the more directly its parameters map to the fundamental characteristics of the system to be simulated. If these characteristics are known, simulation calibration may be unnecessary. Low-level simulations are the norm in networking research, in which many published research results are obtained using packet-level simulators. The parameters of the simulation models pertain mostly to physical characteristics of network links and routers and to configurations of network protocols. It should thus be possible to pick appropriate values for these parameters based on the specification of the target system to simulate. And yet, many authors have found that picking appropriate values is challenging and that calibration is, in fact, necessary for achieving high accuracy [65, 1, 32, 42, 36].

In recent work [17], a simulator of the Pegasus/DAGMan system [24] executing workflows on on both bare-metal and AWS-provided compute resources managed by HTCondor [63] was calibrated. This simulator comes with dozens of configuration parameters. The calibration was performed manually, by comparing simulated executions to several real-world execution logs, finding out information about the platform's characteristics from hardware specifications and micro-benchmark results, and inspecting source code. In the end, the calibrated simulator was shown to achieve high accuracy. While feasible, such manual simulation calibration is extremely labor- and time-intensive. As a result, it is often seen as “witchcraft”. It is poorly documented in the literature and often described vaguely, with statements such as “we carefully calibrated our simulator”, which are ambiguous and widely open to interpretation.

In areas of simulation outside of PDC, and in fact outside of Computer Science, there are a few published examples of automated calibration. These areas are wide ranging, exemplifying the many applications of simulation, and thus of opportunities for simulation calibration. They include everything from flood models [46] to building electrical consumption models [70] to vehicular traffic models [35]. A common theme for all these models is a large number of parameters and some degree of abstraction between the model parameters and their translation to reality. While these models can be calibrated manually by skilled individuals, the above works explore automated calibration to save time and reduce required expertise. The work in this thesis is driven by the same motivation.

To the best of our knowledge automated calibration for PDC simulation, which is the main goal of this thesis, has not been explored. Automation of parameter calibration seeks to solve the issues faced by manual calibration, such as the necessary comprehension of the impact of parameters defining complex abstractions. Automation can be achieved by turning simulation calibration into a black box optimization problem. The input to this problem is one real-world execution trace. The output is a set of simulation parameter values. The objective is for these parameter values, when passed to the simulator, to minimize simulation error, as defined earlier in this section. In the next chapter we present and detail our proposed simulation calibration appraoch.

CHAPTER 3
PROBLEM STATEMENT AND APPROACH

In this chapter we first outline our target problem (Section 3.1). We then describe our our proposed approach, which can be instantiated for arbitrary simulation accuracy metrics and calibration algorithms (Section 3.2). We describe the metrics (Section 3.3) and algorithms (Section 3.4) that we consider in this thesis.

3.1 Problem Statement

Consider a PDC system that includes: (i) a hardware platform; (ii) an application workload to execute; and (iii) a software runtime system that orchestrates the execution of the workload on the platform. Examples include a data center on which a MapReduce job is executed using Hadoop+HDFS, a dedicated HPC cluster on which a tightly-coupled parallel application is executed using MPI, or a batch-scheduled homogeneous cluster on which a workflow application is executed using a Workflow Management System.

A simulator of that same PDC system has been developed using a variety of simulation models, each of which is either custom-developed or provided by a simulation framework. This simulator takes parameters that are used to instantiate all these simulation models.

The real-world execution of the system produces an execution trace, that is, a log of time-stamped execution events and metrics. At a minimum the trace includes an "execution completed" event. The simulator also produces a trace of the simulated execution, implemented so that it is structurally and quantitatively comparable to the real-world trace. Given an accuracy metric that can be computed based on these two traces, the objective is to determine the simulation parameter values that optimize this metric. Simulation calibration is thus an optimization problem whose objective function can be evaluated but is not known analytically.

3.2 Proposed Approach

In this thesis we develop a general, automated simulation calibration approach, the operation of which is depicted in Figure 3.1.

The first step is to collect ground-truth data from a PDC application execution on a real-world hardware/software platform. This real-world execution trace (left part of the figure) is passed to our simulation calibration procedure (center part of the figure). This procedure is instantiated for a particular simulation accuracy metric and with a particular calibration algorithm (top part of the figure). The algorithm repeatedly invokes the simulator with candidate parameter values, thus obtaining a simulated execution trace for which the accuracy metric is computed (bottom part of the figure). In this manner the algorithm searches the parameter space for the parameter values

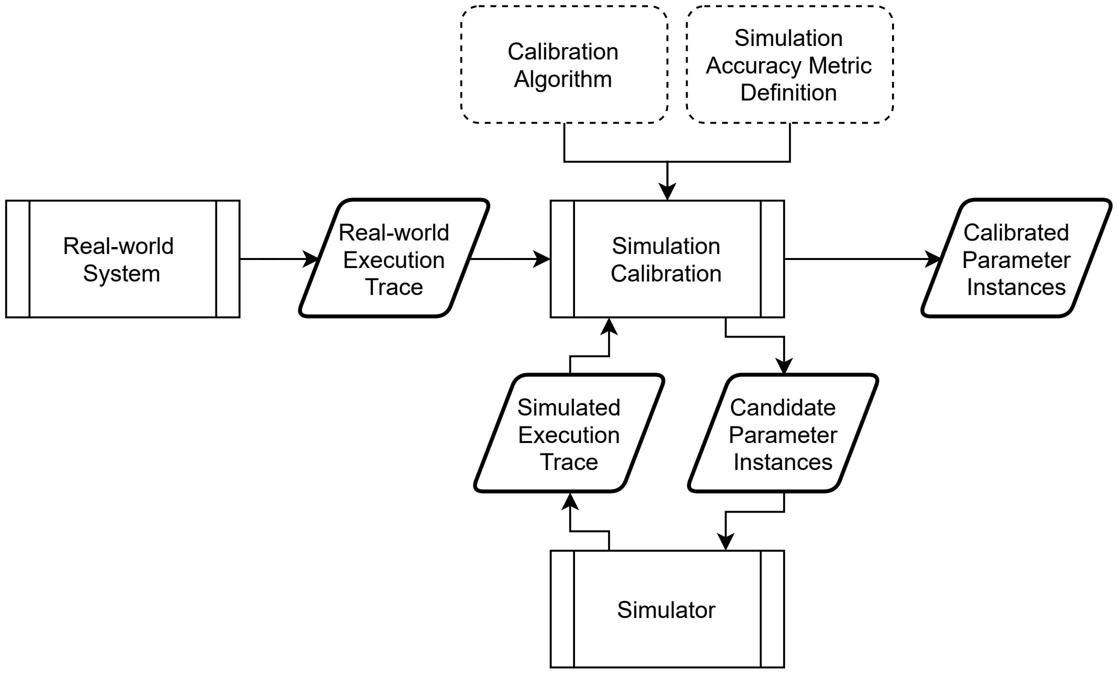
Calibration Algorithm
Simulation Accuracy Metric Definition
Real-world System
Real-world Execution Trace
Simulation Calibration
Calibrated Parameter Instances
Simulated Execution Trace
Candidate Parameter Instances
Simulator

Figure 3.1: Approach for reaching an accurate simulation.

that optimize accuracy. Once this algorithm terminates, the parameter values that achieve the highest accuracy are returned as output (right part of the figure). We say that these values have been calibrated.

We assume that for each parameter the range of its possible values is known. This range is provided by the simulator user, and could be very large. For instance, network bandwidth parameters could be simply set to be between 0 and 1000 GB/sec. The goal is to constrain the search space, which is necessary for some calibration algorithms (e.g., exhaustive search). The narrower the range of a parameter the faster the calibration, but the higher the risk that the best parameter value is outside that range. We also assume that there is a fixed bound on the number of simulator invocations that our approach can place. This is to bound the calibration time. The higher this bound the better the calibrated parameter value.

In this thesis, and in particular in the case study in Chapter 4, we use the accuracy metrics and calibration algorithms described in the next two sections.

3.3 Accuracy Metrics

We consider the two following simulation accuracy metrics:

- **Makespan Error (ME)** – The metric is computed as the absolute value of the difference between the simulated makespan (the total execution time) and the ground-truth makespan obtained from the real-world execution. This metric, which is in seconds, is probably the simplest metric for quantifying the discrepancy between the simulated and the real-world execution. Due to its simplicity, it can also fail to quantify important discrepancies between the simulated and the real-world execution. Specifically, two very dissimilar executions could have the same makespan. As a result, a metric that captures more of the overall structure of the execution could be preferable.

- **Task Completions Mean Squared Error (TCMSE)** – This metrics computes the mean squared error between the sorted workflow task completion times in the simulated and the real-world execution. This metric thus capture some notion of similarity between the patterns of the two executions.

The first metric is the most common accuracy metrics used by most works in the literature. Both metrics quantify accuracy as a single scalar, but the second metric captures information about the temporal structure of this execution. It has been used for manual calibration in previous work [17]. Both metrics in fact quantify simulation error, and thus are to be minimized.

3.4 Calibration Algorithms

Calibration can be done using various methods for searching the parameter space for values that optimize simulation accuracy. We consider the following three primary simplistic search methods and a fourth with a greater degree of complexity:

1. **Exhaustive search** – Given a number of parameters to calibrate, each with a range of values, this algorithm evaluates all parameter combinations by discretizing the parameter space evenly in each parameter range.
2. **Random search** – This algorithm simply evaluates sets of randomly generated parameter values, where each value is sampled uniformly in its parameter range.
3. **Hybrid search** – This algorithm first uses a random search to identify a good set of parameter values (exploration) and then uses an exhaustive search in the neighborhood of those values (exploitation).
4. **Bayesian optimization** – This approach [54, 53] is known to be effective for optimizing black-box functions that are relatively expensive to evaluate. We implement Bayesian optimization using the `BayesianOptimization` function in the `GPyOpt` Python package [4].

Given the dimensionality of the parameter space, there needs to be a bound on the search time to ensure that simulation calibration is tractable. Furthermore, in this work we compare the above algorithms, and to ensure a fair comparison these algorithms should run in roughly the same amount of time. For the first three algorithms above, we simply bound the number of simulator invocation that they can place (to 10,000 invocations). When applying this bound to the Bayesian optimization algorithm, we found that, at least using the implementation in the `GPyOpt` package, the search time was excessively long, much longer than that for the first three algorithms. Using an empirical trial-and-error approach, we opted for configuring the Bayesian optimization algorithm to use 500 “iterations”, which leads to a total execution time that is roughly equivalent to that of the other three algorithms. Consequently, we can perform a fair comparison of our calibration algorithms.

3.5 Conclusion

Although many options exist for the accuracy metrics and calibration algorithm used to instantiate our approach, the options outlined in the previous two sections are sufficient to determine the feasibility of automated PDC simulation calibration. In the next chapter we verify this feasibility via a case study for a production system.

CHAPTER 4
CASE STUDY: CALIBRATING A SIMULATOR OF WORKFLOW EXECUTIONS WITH PEGASUS/HTCONDOR/SLURM ON CHAMELEON

In this chapter we evaluate our proposed approach via a case study. We first describe the production real-world system we use for driving the case study (Section 4.1). We then describe our simulator of this system, which we wish to calibrate (Section 4.2). We identify specific research questions that we wish to answer via this case study (Section 4.3), and then present results pertinent to each question (Section 4.4). Lastly, we summarize our findings (Section 4.5).

4.1 Real-World System

Per the problem statement in Chapter 3, a system consists of: (i) a hardware platform; (ii) an application workflow to execute; and (iii) a software runtime system. In the next three sections we describe each component of the real-world system we use for our case study.

4.1.1 Hardware Platform

We use hardware resources from the Chameleon testbed [20], an NSF-funded experimental platform built to support Computer Sciences systems research. Specifically, our allocation on this testbed consists of **homogeneous dedicated hosts**, with the following hardware specifications:

- Processors: 2x CPU E5-2670 v3 @ 2.30GHz for a total of 48 hardware threads;
- RAM: 16 GB;
- Network: Connected via 10GB Ethernet;
- Disk: HDD SATA 250GB 7200 RPM (model: ST9250610NS).

For most of our experiments the setup consists of four physical hosts, however this is increased to six, ten, and then eighteen hosts for the experiments detailed in Section 4.4.4. In addition, another Chameleon host not included in that number is used strictly for file system access via NFS.

4.1.2 Application Workloads

As motivated in Chapter 1, we have chosen workflows as our target application workloads. Specifically, we consider both *synthetic* and *real-world* workflows. Synthetic workflows, which we have crafted ourselves, are relatively simple with regular structure. Real-world workflows are used to determine how applicable our approach is to large, production workflows.

Synthetic Workflows – We consider 7 synthetic workflow configurations. These configurations vary in their number of tasks and in the dependencies between them. The smallest configuration has a single task, while the largest is a 17-task workflow with many dependencies. Figure 4.1 depicts the smallest three configurations, and other configurations are depicted in Figures A.1-A.4 in Appendix A. All these synthetic workflows are meant to be much simpler than a real-world workflow, and it is relatively easy to examine and analyze their executions. Therefore, not only do they correspond to "easy" cases for simulation calibration, but it is possible for us to manually inspect their execution logs (simulated and real-world) to investigate causes of simulation error.

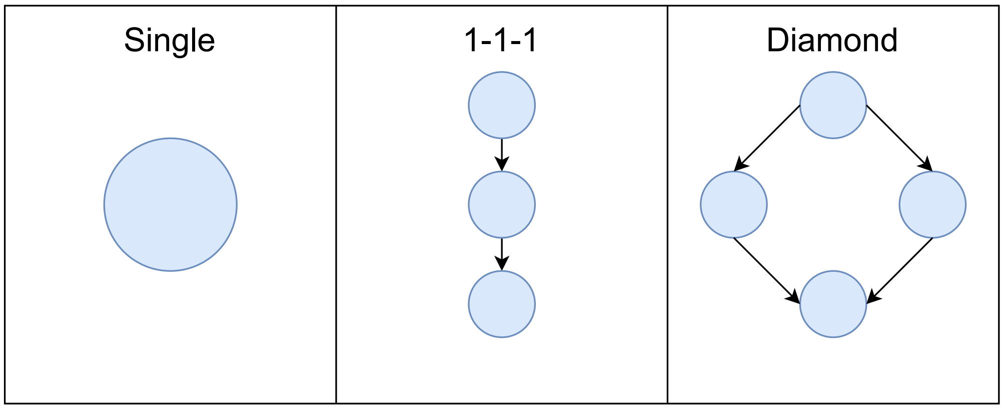

Figure 4.1: Structure of our first three synthetic workflows (other workflows are depicted in Appendix A.)

Real-world Workflows – We have selected representative workflow configurations for two well-known scientific applications: Montage and Epigenomics. Montage is an application that performs astronomical image processing, whereas Epigenomics is a bioinformatics application that converts, filters, maps and merges gene sequences into a single map. These workflows are executed in production using the Pegasus system [24], and actual workflow configurations are available from the WorkflowHub project [29] project. Specifically, in this case study we use a 58-task Montage configuration and a 71-task Epigenomics configuration. Figure 4.2 depicts a small Montage configuration, and a small Epigenomics configuration is also depicted in Figure A.5 in Appendix A.

4.1.3 Software Runtime System

We use a production software runtime system for executing workflows on Chameleon. Namely, we use the Pegasus Workflow Management System, which orchestrates workflow execution by submitting jobs to the HTCondor High Throughput Computing (HTC) system, which itself submits jobs to the Slurm batch-scheduling RJMS. This runtime system, and its deployment on Chameleon, is representative of complex cyberinfrastructure deployments used to execute scientific applications [13]. Importantly, the use of Pegasus, HTCondor, and Slurm together entails many software interactions and intra- and inter-host communications, which lead to various and complex overhead

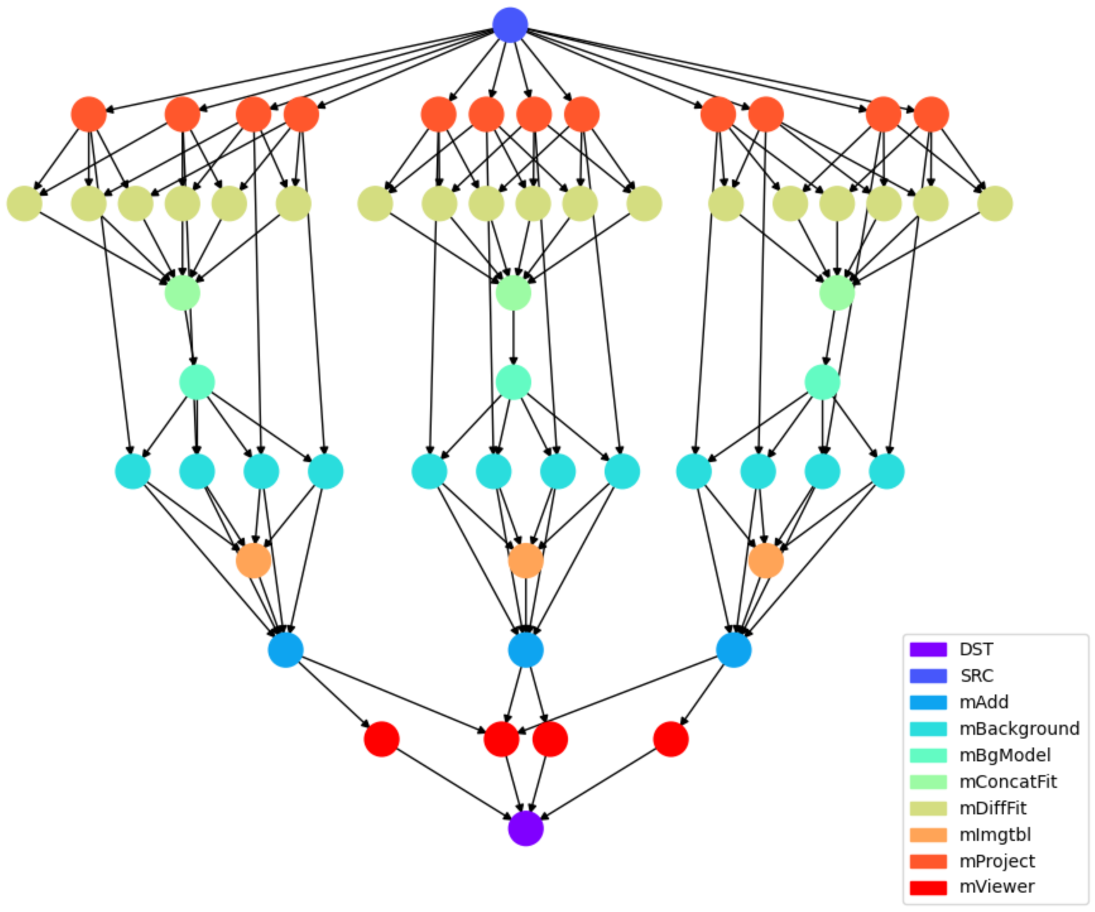

Figure 4.2: Sample Montage workflow.

behaviors (which hopefully can be captured by a well-calibrated simulator). In what follows, we describe each of the three components of this runtime system.

Pegasus

Our workflows are executed using the Pegasus Workflow Management System [25]. Pegasus is being used in production to execute workflows for dozens of high-profile applications in a wide range of scientific domains, as seen in the project's "Application Showcase" [1]. Pegasus provides the necessary abstractions for scientists to create workflows and allows for transparent execution of these workflows on a range of compute platforms including clusters, clouds, and national cyberinfrastructures. During execution, Pegasus translates an abstract resource-independent workflow into an executable workflow, determining the specific executables, data, and computational resources required for the

[1] `https://pegasus.isi.edu/application-showcase/`

execution. Workflow execution with Pegasus includes data management, monitoring, and failure handling. In particular, workflow execution is handled via the HTCondor HTC system (described hereafter). Pegasus binds the execution of a each workflow task to a particular compute resource at the onset of the workflow execution and then strives to execute that task on that resource (unless the resource experiences a failure). Workflow executions with Pegasus experience relatively high overhead, as clearly seen in workflow execution logs. It turns out that this overhead is due to the way in which the interaction between Pegasus and HTCondor is implemented. Specifically, rather than relying on a *push model* for communicating execution events (such as task completions, resources becoming idle), instead a *pull model* is used. Fixed polling periods are used (on the order of 30 seconds) is used for pulling updated task and/or resource status updates. As a result, large delays can be experienced, as well as propagated throughout the whole system.

HTCondor

As mentioned earlier, Pegasus executes tasks via HTCondor [63] (more specifically, using HTCondor's DAGMan component [30] and HTCondor's "grid universe" feature). HTCondor is a HTC system that is designed to execute large numbers of compute jobs on a range of compute platforms. It is primarily used for serial jobs, although it has been extended to support parallel jobs as well. In our case study, all jobs are sequential (i.e., one job per workflow task). In the standard use of HTCondor, so called "HTCondor worker" daemons are started on compute resources. These daemons then make it possible to execute HTCondor jobs on those resources. In our use case, instead, the jobs are tagged with the "grid universe" attribute. As defined by HTCondor, jobs with this attribute are to be redirected to a compute platform, typically a cluster, managed by a batch scheduler (such as Slurm). The goal is to execute these jobs, if possible, on any idle core available on that platform at the time of job submission. DAGMan (Directed Acyclic Graph Manager) is a high-level scheduler built into HTCondor that explicitly manages inter-task dependencies. It is thus useful for execute workflow applications, which is why Pegasus relies on DAGMan. DAGMan maintains a current list of those jobs that are "ready", i.e., whose parent jobs have already completed, so as to ensure that jobs are executed in the correct order. HTCondor reports idle compute resources to DAGMan, which then picks one of the ready jobs for execution on these resources. To achieve all the above, HTCondor consists of multiple daemons (master, collector, negotiator, schedd, etc.) that coordinate and work together to orchestrate and monitor job executions. The intercommunication of these daemons with each other and with other components of the software infrastructure (e.g., the batch scheduler), is one overhead in the workflow execution. Furthermore, there are some artificially introduced overhead in the system. In particular, we found that DAGMan includes a fixed sleep (of 5 seconds) for each job submission.

Slurm

We consider compute nodes managed via the popular and ubiquitous Slurm batch scheduler [71]. Users (in our case HTCondor) submit jobs to a batch scheduler requesting some number of compute nodes for some length of times. These jobs requests are placed in a queue (the so-called "batch queue"), and a particular scheduling algorithm is used to select which of these jobs in the queue is started on available, idle compute nodes. The typical algorithm, which is implemented by default in Slurm, is first-come-first-serve with backfilling. Backfilling is a technique that allows small/short jobs to jump ahead in the queue, which is typically deemed desirable to increase compute node utilization [49]. Slurm, much like HTCondor, consists of multiple daemons responsible for enabling and managing job executions. The main such daemons are a a control daemon, which runs on a "head node", an optional database daemon, and a `slurmd` daemon that runs on each compute node and reports to the control daemon. Here again, inter-daemon communications add overhead to the workflow execution.

Deployment on Chameleon

We consider a small-scale deployment of this system on the Chameleon testbed. Figure 4.3 depicts this deployment when two hosts are used as compute nodes managed by Slurm. Another host executes the Slurm coordinator, and another executes the HTCondor coordinator. We instantiated this deployment for 2, 4, 8, and 16 compute nodes. Therefore we only consider relatively small-scale deployments. This is due to limited testbed resources and competition for these resources when requiring them, as we do, for long periods of contiguous time.

4.2 Simulator

We have implemented a simulator of the system described in the previous section. This simulator is implemented in C++ and consists of approximately Ï 800 LOCs. This simulator is built on top of the WRENCH simulation framework [19, 69], which provides high-level simulation abstractions for implementing simulators of complex PDC systems. In particular, WRENCH supports both HTCondor and batch schedulers, such as Slurm, natively. Therefore, the development effort for our simulator was minimal. Instead, most of our simulation development effort was devoted to extending WRENCH so that it support HTCondor's grid universe feature. Although we performed this development specifically for this case study, it is now part of the WRENCH code base and available to all WRENCH users who wish to easily simulate HTCondor, including its grid universe feature. Recall that WRENCH itself builds on top of SimGrid [16], a popular accurate and scalable simulation framework that has been used to obtain simulation results in 500+ research publications. As discussed in Chapter 2, WRENCH and SimGrid make it possible to implement simulators that can be extremely accurate, *provided they are well-calibrated.*

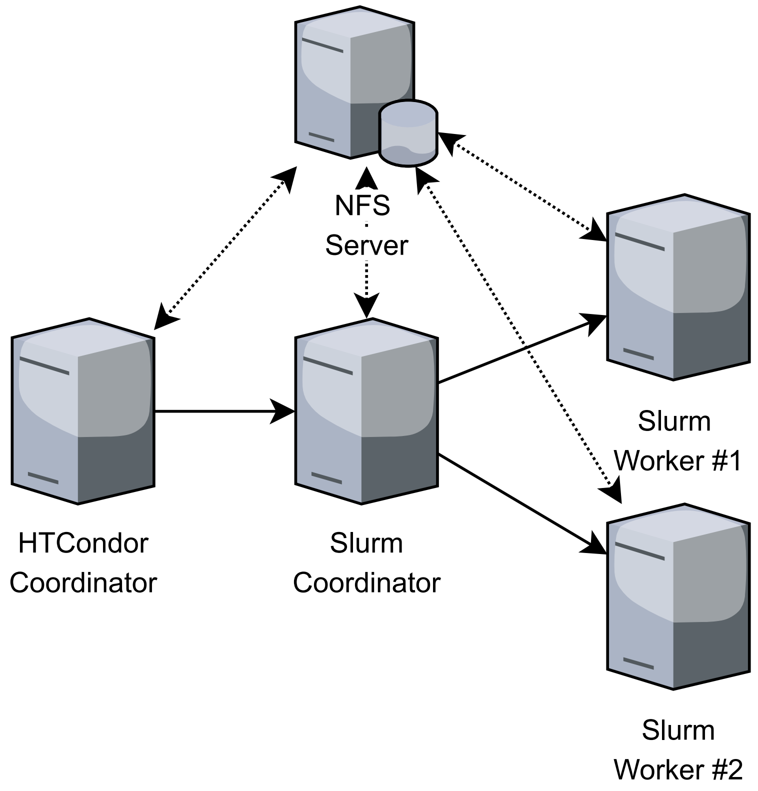

Figure 4.3: Hardware resources in the Chameleon testbed when using two compute nodes (Slurm workers).

The simulation models within our simulator are configured via 5 distinct parameters, as follows (ranges used for each parameter can be found in square brackets):

- Disk Bandwidth – [1,400] The effective disk data transfer rate (R/W), in MBps. It is used to simulate each host's primary disk that is used to store workflow data.
- Link Bandwidth – [1,1000] The effective bandwidth between hosts, in MBps.
- Pre-Execution Overhead – [1,2000] An overhead in seconds that causes a delay before the execution of the first workflow task. This overhead is significant in real-world executions and is due to interactions between Pegasus/HTCondor/Slurm as well as idiosyncrasies of some of these systems (as explained in Section 4.1.3).
- Post-Execution Overhead – [1,2000] An overhead in seconds that causes a delay experienced after the execution of each level of the workflow. This overhead is again significant in real-world executions for the reasons stated above (see Section 4.1.3).
- Flop rate – [250000000,3000000000] The effective computational speed, in floating point operations per second, of the compute nodes' cores.

Our objective is to apply our proposed automated simulation calibration approach to compute values for all the above parameters that maximize simulation accuracy.

Our simulator is available on GitHub [2].

4.3 Research Questions

The purpose of this case study is to answer the broad research question: **Is automated calibration of PDC simulators feasible and effective to maximize simulation accuracy?** To do so, we start by answering the following more specific research questions:

- *Research Question #0: Is simulation calibration necessary in the first place?* As discussed in Chapter 2, it is not clear how often simulation calibration is performed in the literature. Anecdotal evidence seems to indicate that many simulator users tend to pick simulation parameter values based on their own knowledge (or best guesses) regarding the target system. In the scope of this case study, we wish to quantify the loss in simulation accuracy that such a, seemingly reasonable, approach can cause.

- *Research Question #1: What are sufficient simulation accuracy metrics?* Simple metrics that capture only one aspect of the simulation outcome can be easily implemented and understood (e.g., overall execution time), but there are also more sophisticated metrics to consider. Metrics that capture the granular temporal structure of the simulation outcome may lead to preferable, more generalizable parameter instantiations. In Section 3.3 we have proposed two metrics, one simple and one more sophisticated. In the scope of this case study, regardless of the calibration algorithm in use, we wish to quantify the difference in overall simulation accuracy when calibrating with either one of these metrics.

- *Research Question #2: What are effective algorithms for searching the parameter space for parameter instantiations that maximize simulation accuracy?* A range of algorithmic options can be envisioned for the simulation calibration process, and we have outlined several options in Section 3.4. Some are brute-force algorithms (such as exhaustive searches) that may have difficulties exploring a multi-dimensional parameter space thoroughly in reasonable amounts of time. Others (such as random searches or Bayesian optimization) should fare much better. In the scope of this case study, we wish to compare the effectiveness of these different algorithmic options.

- *Research Question #3: How generalizable are automatically computed calibrations?* After a calibration is computed based on a particular real-world execution trace, is it unclear to which extent this calibration remains valid when used to simulate different execution scenarios (e.g.,

[2]`https://github.com/wrench-project/automated-calibration`

different application workflow characteristics, different platform characteristics and scales). Ideally, the calibration could be performed with a simple (small-scale, quick-running) real-world experimental scenario, and still lead to high accuracy, or "generalize", to a range of other scenarios, and in particular larger-scale scenarios. In the scope of this case study we wish to quantify the extent to which computed parameter calibrations are generalizable beyond the specific scenario used as input to the calibration process.

4.4 Results

In all that follows, we are quantifying the accuracy of the calibrated simulator using the absolute value of the difference, in percentage, between the simulated makespan and the observed real-world makespan. Note that the calibration may have been computed using this very metric, ME, or the TCMSE metric, as defined in Section 3.3. We use the ME for assessing the accuracy of the calibrated simulator so as to ensure consistent comparisons between methods, and also because it is the main metric used in the literature.

One factor to keep in mind when examining our results is the variability of real-world executions. From our experience with Chameleon, the standard deviation of a workflow makespan was found to be 6 to 7%. This is because there are always dynamic factors that perturb executions. This is particularly the case in a shared cloud environment such as Chameleon, where other users can cause unexpected load on the system. While our compute nodes are dedicated to our purpose, the network infrastructure is not. Network load can change drastically from hour to hour and impact file transfer times. One possible way to mitigate the variability of the execution is to reserve host in the same rack of the physical infrastructure, which is in principle feasible with Chameleon. However, we found that doing so is not always possible or practical.

4.4.1 Research Question #0

The first question to address is whether automated calibration is necessary to begin with. Or, in other terms, what is the effectiveness of simply picking parameter values based on (hopefully educated) guesses? Recall that, based on the existing literature, this approach is by far the most common. For this case study, we know the hardware specifications of our hosts as stated on the Chameleon Web site (see Section 4.1.1). Since from these specifications we know the model and number of the disks, we can also find out their advertised data transfer rates (which are 115 MBps). More difficult to estimate is the network bandwidth, especially since network contention occurs on the platform and the network configuration depends on the racks on which allocated hosts are located. The Chameleon Web site provides very little information, and so a reasonable estimate based on our own experience is 2500 MBps. Determining flop rates is also difficult, but given that we know CPU information we are able to find published benchmark results, which lead us to use 1.150 billion flop/sec. The last two parameters which represent overhead added to the execution by Pegasus/HTCondor/Slurm are more difficult to pick. Based on our observations of a few executions, setting both of them to a few minutes (180 seconds) each seems reasonable. While more effort could perhaps be devote to picking better values, the above is representative of what is typically done in the PDC literature when calibration is not attempted.

Using these manually calibrated parameters we run a simulation of our fourth synthetic workflow, which features an eight-task harpoon-join configuration. We find that the simulation error is 36.243%. And yet, as seen in upcoming sections, it is possible to compute parameter values that achieve almost perfect accuracy. This illustrates the drawback of not performing simulation calibration. And still, this is only for picking 5 parameter values, each of which seemingly directly maps to key properties of the target system (or in fact are provided as known specifications). Given the observed simulation error, we could then embark on a manual simulation calibration effort, i.e., "tweaking" parameter values until the simulation error is acceptable. That tweaking by trial-and-error is precisely what our proposed approach seeks to automate.

4.4.2 Research Question #1

In Section 3.3 we have defined two metrics can can be used by our proposed automated simulation calibration approach: the simple ME (Makespan Error) metric and the more complex TCMSE metric (Task Completions Mean Square Error). The question is whether using the TCMSE metric is advantageous w.r.t using the ME metric (see discussion in Section 3.3).

Table 4.1 shows, for each of our 9 workflows when executed on 2 compute nodes, the simulation error when using the two metrics for executing the calibration algorithm. These results are obtained when using the Exhaustive Search calibration algorithm.

Table 4.1: Simulation error for both calibration metrics when using the Exhaustive Search calibration algorithm.

	Calibration Metric	
Workflow	ME	TCMSE
SingleTask	8.63%	8.63%
1-1-1	0.00%	0.00%
Diamond	0.07%	9.42%
Harpoon	0.40%	1.38%
1-10-1	0.70%	2.55%
1-5-1-5-1	0.46%	7.30%
1-3-(*4)-1	8.11%	8.11%
Epigenomics	0.20%	4.70%
Montage	0.22%	2.77%

From the results in the table, and perhaps counter-intuitively, we see that calibrating using the more complex TCMSE metric in fact leads to higher error than calibrating with the ME metric. For other calibration algorithms, such as Bayesian optimization, the two metrics lead to similar results. However, in none of our results does using the TCMSE metric lead to a significant improvement.

It is possible that for more complex settings, i.e., outside the scope of this case study, using the TCMSE metrics would bring some benefits (since after all it does capture features of the temporal structure of the execution). This said, in all that follows, unless specified otherwise, we discuss results obtained when using the ME metric for calibration.

Full results, including all results when using the TCMSE metric for calibration, are available in Appendix B.

4.4.3 Research Question #2

Table 4.2: Simulation error for each calibration algorithm when using ME metric.

	Calibration Algorithm				Manual Calibration
Workflow	Exhaustive	Random	Hybrid	Bayesian	
SingleTask	8.63%	0.00%	0.09%	0.00%	6.47%
1-1-1	0.00%	0.00%	0.09%	0.39%	20.70%
Diamond	0.07%	0.00%	0.12%	0.02%	38.04%
Harpoon	0.40%	0.00%	0.28%	0.13%	36.23%
1-10-1	0.70%	0.02%	0.20%	0.09%	51.28%
1-5-1-5-1	0.46%	0.01%	0.05%	0.10%	42.14%
1-3-(*4)-1	8.11%	0.02%	0.02%	0.07%	51.31%
Epigenomics	0.20%	0.02%	0.03%	0.20%	61.35%
Montage	0.22%	0.02%	0.59%	0.22%	52.66%

Table 4.2 shows simulation error for all our workflows, when executed on 2 compute nodes, and our four calibration algorithms. Actual parameter values computed by these algorithms are available in Tables B.1-B.9 in Appendix B.

The main observation from these results is that all the algorithms perform well. However, there is some degree of separation. The Exhaustive Search algorithm performs the worse, the only method with any error over 1%. The Bayesian Optimization algorithm and the Hybrid Search algorithm both perform well, but the clear standout is the Random Search algorithm. Even though we limit the number of simulation invocations to 10,000, it is able to achieve near perfect calibration, i.e., near-zero error, for every workflow. We expect that for a more “difficult” simulation calibration problem (i.e., more parameter dimensions) the algorithms would exhibit more differences. In particular, we would expect the Bayesian Optimization algorithm to fare increasingly better as dimensionality increases.

The difference between Exhaustive and Random is somewhat surprising, as we would expect these to achieve similar results. As the number of simulation invocations is limited, recall that Exhaustive is a grid search, i.e., it discretizes the parameter search space. In our experiments, this means that Exhaustive only evaluates six different values for each parameter. One likely cause of the poorer performance of Exhaustive is that the five simulation parameters are not all created equal: they can have varying degrees of impact on the simulated execution time. This is due to factors like bottle necking between disk speed and link speed (if one of the two is a clear bottleneck, the other parameter does not really matter), and, for similar reasons, the composition of the workflows (computation vs. I/O). As Exhaustive only evaluates six values for each parameter, it is likely that no good value is found, and having a poor value for an impactful parameter results in the poor simulation accuracy overall.

One caveat in our results is that they are obtained for a single real-world execution trace. Yet, as mentioned previously, real-world executions experience some level of platform "noise", such that from our executions we experienced a standard deviation of 6-7% in makespan, for the same workflow execution. This means the calibrations computed by our algorithms are overfitted to optimize for one particular data point. As a result, calibration leads to accurate results for this specific workflow execution, but may not be accurate for a different execution of that same workflow. Given additional time and resources to generate a large sample of real-world execution traces, the way to avoid overfitting would be to calibrate over that large sample, using a standard training-testing set approach. The one problem with this approach is that generating that many real-world traces requires a lot of time and access to the computing environment. We were not able to obtain this data for our particular testbed. Also, from a more general perspective, obtaining a very large sample may be impractical give the resource expense. For the purposes of our case study, unfortunately, limited testbed resources and time, along with recurrent technical issues, we have opted for using a single real-world execution trace. As a result, it is important to keep the overfitting aspect of our results in mind.

Given the results in this section, in all that follows we only present results obtained using the Random Search calibration algorithm.

4.4.4 Research Question #3

PDC simulators are typically used to simulate scenarios that cannot be executed in the real world, in particular, scenarios for other application workloads and for other platform configurations. In this section, we seek to quantify the extent to which a simulation calibration computed based on a particular real-world scenario can be reused for other scenarios with some degree of accuracy. We first look at using a calibrated simulator for different workflows and then for different platform scales.

Different Workflows

Table 4.3: Percent simulation error when using Random Search with the ME metric.

Random / ME	Single Task	1-1-1	Diamond	Harpoon Join	1-10-1	1-5-1-5-1	1-3-(*4)-1	Epigenomics	Montage
Single Task		8.53%	38.69%	47.25%	60.11%	51.59%	62.34%	69.14%	80.60%
1-1-1			18.67%	31.38%	49.61%	44.12%	52.85%	61.83%	51.56%
Diamond				5.63%	30.93%	18.13%	32.78%	36.36%	21.46%
Harpoon Join					33.16%	11.40%	38.40%	46.70%	78.27%
1-10-1						27.25%	6.32%	12.86%	4.64%
1-5-1-5-1							27.08%	13.43%	33.79%
1-3-(*4)-1								11.81%	26.33%
Epigenomics									5.85%
Montage									

Table 4.3 shows simulation error observed when using the parameters calibrated for one workflow configuration to simulate the execution of another workflow configuration. More precisely, the rows indicate which workflow was used for computing the calibration and the columns indicate which workflow executions are simulated using this calibration. For instance, the cell at row "Diamond" and column "1-5-1-5-1" shows the simulation error observed for simulating the execution of the 1-5-1-5-1 workflow when the simulator is calibrated based on the real-world execution of the Diamond workflow. Note that the numbers on the diagonal (omitted from figure) would be the same numbers as those in the second column of Table 4.2.

The results in Table 4.3 show that, as expected, there is a loss in accuracy when applying a calibration computed for one workflow to another (larger) workflow. This loss can be quite high. For instance, calibrating with the synthetic Harpoon workflow leads to 78.27% error when simulating the production Montage workflow. In some cases, the results are not as poor. For instance, using the calibration computed based on the execution of the 1-10-1 workflow leads to a much lower 12.86% error for Epigenomics and 4.64% error for Montage. These margins of error are definitely acceptable to most users (as they are much better than what is often reported in the literature, and better than what can be seen above in Section 4.4.1). Recall also that our workflow execution suffers from variability due to platform noise, which likely increases simulation error in these results. This said, we have no reasonably hypothesis regarding why some results in Table 4.3 are better than others. For instance, we have no explanation for why using the 1-10-1 calibration for simulating Epigenomics leads to high accuracy, but not using the 1-5-1-5-1 calibration.

Overall, the results in Table 4.3 are disappointing, and seem to indicate that automatically computed calibrations do not generalize to other workflow scenarios, or at least not consistently if high accuracy is needed. As explained in Section 4.4.3, part of the lack of accuracy can likely be attributed to the overfitting in our calibration process caused by using just a single real-world execution trace. In an attempt to understand any other reasons for these results we have examined simulated and real-world execution logs. We found that the culprit for these poor results is the calibration of the "overhead" simulation parameters (see Section 4.2). This is because the calibration algorithm can find a good solution for a specific scenario, but that solution over- and under-estimates various overheads. When using the computed calibration, the over- and under-estimates compound and do not scale, leading to poor accuracy. Furthermore, we find that our simulator does not fully simulate all overhead behaviors of the real-world system, in which case there are inherent limits to what calibration can achieve for that simulator. We believe that, while some overhead in the real-world system can be attributed to behaviors of HTCondor and Slurm, the largest portion is due to Pegasus.

To further verify the above explanation, we conducted experiments without using the Pegasus workflow management system, which is responsible for the largest amount of overhead. We thus configure our simulator to not simulate these overhead (i.e., we set them to 0 seconds). Due to not using Pegasus, we are not able to execute the Epigenomics and Montage workflows. Table 4.4 is similar to Table 4.3, but shows results only for the synthetic workflows executed without Pegasus (i.e., via custom scripts).

Table 4.4: Percent simulation error when using Random search with ME metric on Non-Pegasus Synthetic Workflows.

Non-Pegasus Random / ME	Single Task	1-1-1	Diamond	Harpoon Join	1-10-1	1-5-1-5-1	1-3-(*4)-1
Single Task		6.31%	17.39%	11.10%	22.17%	18.77%	20.14%
1-1-1			21.87%	15.22%	25.33%	22.17%	23.22%
Diamond				7.37%	7.67%	2.20%	5.25%
Harpoon Join					13.28%	8.47%	10.69%
1-10-1						6.58%	3.03%
1-5-1-5-1							2.83%
1-3-(*4)-1							

Results in Table 4.4 are significantly better than those in Table 4.3. These results show that calibrations computed based on a given workflow execution do generalize to other workflows. These results, albeit in a limited setting, are encouraging but point to many interesting questions. The first question is that of the inherent accuracy of a simulator, as we have seen that our simulator fails to capture the complex overhead behavior of our real-world system. Another, perhaps more interesting, question, is what set of workflow configurations should be executed so as to obtain simulation calibration that can be re-used with high confidence for large workflows. This question it outside the scope of this thesis but provides fertile grounds for future work.

Different Platform Scales

In the previous section we attempted to use a calibrated simulator for a particular workflow execution to simulate the execution of another workflow. In this section we now simulate the execution of the same workflow, but using a larger number of compute nodes. The question is whether a calibration computed for some platform scale can produce reasonable accuracy when simulating a different platform scale.

Table 4.5: Simulation error when increasing the number of compute nodes for Montage and Epigenomics.

	Simulation error	
# compute nodes	Montage	Epigenomics
2	0.02%	0.02%
4	23.69%	7.35%
8	22.92%	29.87%
16	41.21%	18.15%

On our testbed we executed the Montage and Epigenomics workflows on 2, 4, 8, and 16 nodes. We used the 2-node execution to calibrate the simulator for each workflow. Table 4.5 shows simulation error for both workflows under these conditions.

The results in the table show that, as expected, simulation error increases when using the simulator calibrated for the 2-node executions to simulate executions on more nodes. This increase in simulation error is not monotonic, and the simulation error tends to be larger for Montage that for Epigenomics. As in the previous section, the results are somewhat disappointing, even though a ∼20% simulation error is commonly seen in the literature, and is still less than our manually calibrated results from Section 4.4.1. Here again, and as mentioned in Section 4.4.3, the lack of accuracy can partially be attributed to the overfitting in our calibration process caused by using a single real-world execution trace to compute the calibration.

Inspecting the discrepancies between the simulated and the real-world executions, we find that with higher numbers of compute nodes, the workflow execution time tends to plateau. For instance, the real-world Montage execution takes roughly the same amount of time on 8 and on 16 nodes. However, the workflow parallelism should make it possible for the execution to be faster on 16 nodes. This expected behavior is seen in the simulated execution, but not in the real-world execution. This is due to Pegasus suffering from overheads that our simulator fails to implement. This is essentially the same explanation as in the previous section: our simulator does not sufficiently accurately simulate Pegasus. On a side note, these overheads could be considered flaws of Pegasus, which could be remedied by its developers (and that our simulator made it possible to discover!).

4.5 Conclusion

In this chapter we have applied our proposed automated simulation calibration approach to a particular case study for a production system. Our main findings are:

- Picking simulation parameter values based on knowledge of or best guesses for the characteristics and behavior of a real-world PDC system does not lead to high simulation accuracy.

This was demonstrated for this case study, even though the number of simulation parameters is relatively small and these parameters seem to have a direct mapping to features of the real-world system. As a result, a systematic, automated simulation calibration approach is needed.

- We experimented with two simulation accuracy metrics to drive the automated simulation calibration procedure: Makespan Error (ME) and Task Completions Mean Square Error (TCMSE). The former is simpler, while the latter captures some details of the temporal structure of the execution. Within the purview of our case study, the simpler ME metric leads to better results. Note, however, that in all our results we quantify simulation accuracy after the fact using this same metric, as done in the literature. Therefore, it is perhaps not surprising that using that very same metric for computing the calibration leads to good results. One could hope that calibration computed using the TCMSE metric would be more accurate or more generalizable, but we have not observed this in our case study.

- We experimented with four different calibration algorithms. Within the purview of this case study, Random Search, Hybrid Random/Exhaustive Search, and Bayesian Optimization were able to outperform Exhaustive Search when calibrating on a single execution trace. All three of the former were able to reduce ME and TCMSE to low values under these circumstances. Exhaustive likely leads to the worst results as it cannot search the relatively large parameter space effectively and is most vulnerable to differences in impact (on makespan) between the varying parameters. We note that the single execution trace used to calibrate will introduce additional error when attempting to use the computed calibration to simulate other executions than that used to obtain that particular trace.

- We experimented with using a simulator calibrated for a specific workflow execution trace to simulate another workflow (e.g., a different application workload, a large execution platform). We found that simulation error increases (from a few percent to about 20%) in most cases. The error is still better than results obtained via manual calibration. We identified that complex overhead behavior of our target system, namely that of the Pegasus Workflow Management System, as the main source of error. This is caused by the simulator not fully capturing that overhead behavior (some of which is undesirable not nevertheless present). As a result, computed calibrations, when applied to different workflows or platform scenarios, do not lead to low simulation error. We have verified that when removing this overhead behavior (i.e., executing workflows without Pegasus), our computed calibrations are more accurate and generalizable.

CHAPTER 5 CONCLUSION

In this final chapter we provide a brief summary of our contributions and highlight possible directions for future work.

5.1 Summary of Contributions

In this thesis, we have proposed, developed and evaluated an automated simulation calibration approach for PDC simulators. Specifically, our contributions are:

1. We have designed a generic automated simulation calibration approach, which can be instantiated for arbitrary simulation accuracy metrics and calibration algorithms (i.e., algorithms that search the simulation parameter space in order to maximize the accuracy metric).

2. We have instantiated our approach for two simulation accuracy metric definitions: (i) a simple metric that captures only the overall application execution time; and (ii) a more complex metric that captures the temporal structure of that execution. We have also instantiated our approach for four calibration algorithms: exhaustive search, random search, hybrid random/exhaustive search, and Bayesian optimization.

3. We have built a simulator of a complex real-world, production system, namely a system that uses the Pegasus Workflow Management System [24] to execute various workflow applications on compute nodes managed by the Slurm [71] batch scheduler and accessed via HTCondor [62], all deployed on the Chameleon Cloud testbed [37]. This particular system configuration is used in real-world production settings [13].

4. We have collected real-world execution data both for synthetic and production workflow applications on this system.

5. We have applied our simulation calibration approach to our simulator of this system and have be able to quantify the extent to which it is successful in optimizing simulation accuracy.

6. Our key finding from the above activities is that simulation calibration of PDC simulators is a promising approach for increasing simulation accuracy, even though challenges remain. Specifically:

    - Simply picking simulation parameter values based on knowledge of the target PDC system is difficult, and an automated simulation calibration approach is needed.
    - While several options are possible for the simulation accuracy metric used to compute the calibration, in our case study we found that the simple, most common metric (makespan error) leads to good results.

- We experimented with four different calibration algorithms in our case study. When parameters are calibrated to a single trace, Random Search, Hybrid Search, and Bayesian Optimization algorithms are all able to reduce the simulation error to low values (below 1%), with the Random Search algorithm fitting the training data better than the other algorithms. All these algorithms lead to better results than the Exhaustive Search algorithm. Whether this relative advantage generalizes to new data remains to be tested.
- When using a simulator calibrated for a specific workflow execution trace to simulate another workflow, the calibration being computed using the Random Search algorithm, we found in our case study that the calibration generalizes in that it still leads to results better than manual calibration (as seen by comparing the percentages in Table 4.3 to the the last column of Table 4.4.3). We found, however, that simulation error increases significantly. This increase is due to overfitting to a single workflow execution trace during the calibration process, as well as to the fact that our simulator fails to capture complex (and undesirable but nevertheless present) overhead behaviors of our target production system, and in particular that of Pegasus. We have confirmed that this second source of error is prevalent by running experiments without Pegasus.

5.2 Future Work

This work is only a first step toward an automated calibration solution for PDC simulators. As such, several promising future research directions can be envisioned:

- Although our results showed that it is possible to compute automatically an accurate simulation calibration for a specific execution trace. This calibration can generalize to other scenarios to some extent, leading to results better than manual calibration (ass in by comparing results in Table 4.3 to the manual calibration results in Table 4.4.3). Nevertheless, the simulation error remains relatively high. One difficulty here is that the simulator of the real-world system may not capture all relevant behaviors of that system precisely. In our case study, we found that the overhead behavior of the real-world system was more complex that simulated by the simulator. As a result, although it was possible to calibrate this simulator for each individual execution trace, these calibrations under- and over-estimate overheads to “make it fit”. But these overhead values are then not applicable to other scenarios, or at least not in a way that provides consistently high simulation accuracy. There is thus an interesting tension between quality of the simulator itself and the quality of the computed calibration. A crucial future work direction is to explore this tension. This exploration will require the development of different versions of the simulator, where each version captures the target system’s behavior with a different degree of faithfulness.

- The case study used in this thesis is representative of production PDC systems and application workloads, but is only one of many possible options. It will be necessary to conduct other case-studies with different real-world systems and simulators thereof. This is so as to verify that our approach is generally applicable beyond the scope of the case study in Chapter 4. In particular, it is crucial to evaluate our different calibration algorithms for simulators that require even more simulation parameters. We expect that algorithms such as Bayesian optimization will perform better in higher dimensions. But increasing the number of dimensions will also likely motivate the need for exploring alternate calibration algorithms.

- A natural next step is to release our simulation calibration approach as a Python package re-usable by others. This would include tools to collect real-world execution traces, compute simulation accuracy metrics, and compute calibrated parameter values.

- The key motivation for this research is to automate simulation calibration because manual simulation calibration is not tractable. But regardless of how easy our approach renders the simulation calibration process, this process still requires access to a real-world system deployment. In some cases, users may not have access to such a deployment. In other cases, they only have access to systems that are insufficient for or too dissimilar from their targeted simulated scenario. What is needed instead is a community catalog of simulation calibrations. Specifically, developers and users of particular simulators could, using our proposed approach, easily calibrate their simulators for any number of production system. The simulator software could then be published along with sets of computed calibrations (i.e., sets of parameter values for particular system specifications) so that others can use these simulators with high confidence. This would enable a drastic improvement in the scientific methods used for simulation-based research in the PDC field.

APPENDIX A
APPENDIX FOR SECTION 4.1.2

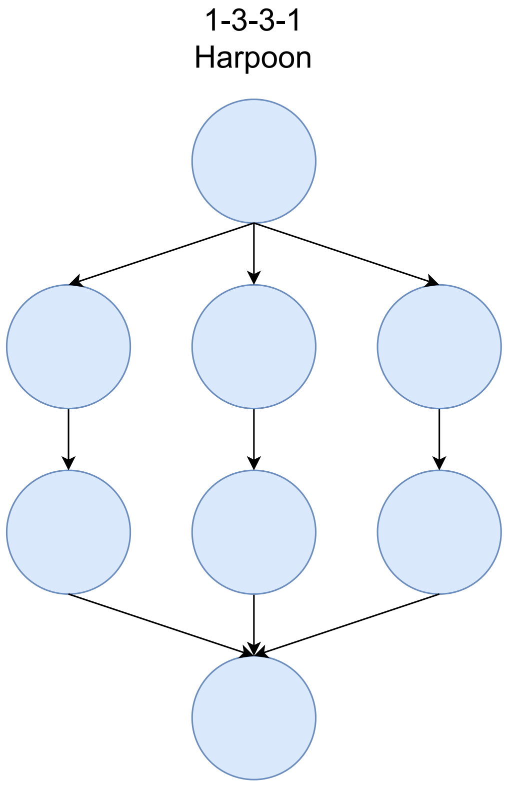

Figure A.1: Structure of Harpoon-Join synthetic workflow.

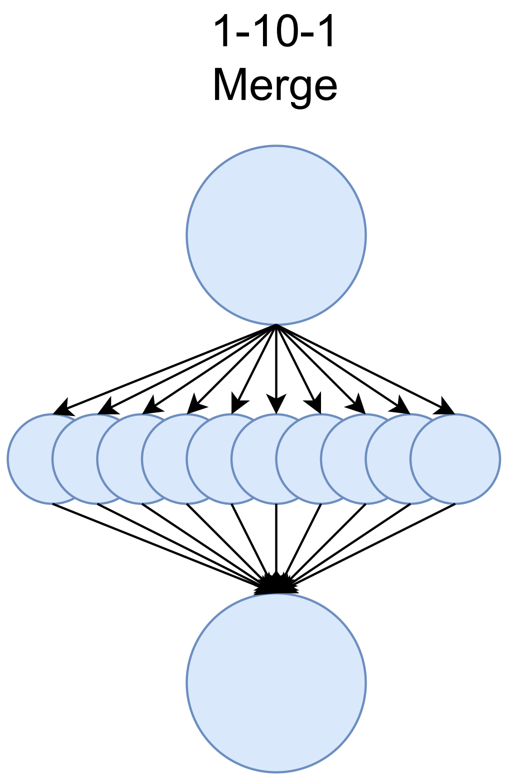

Figure A.2: Structure of 1-10-1 Join synthetic workflow.

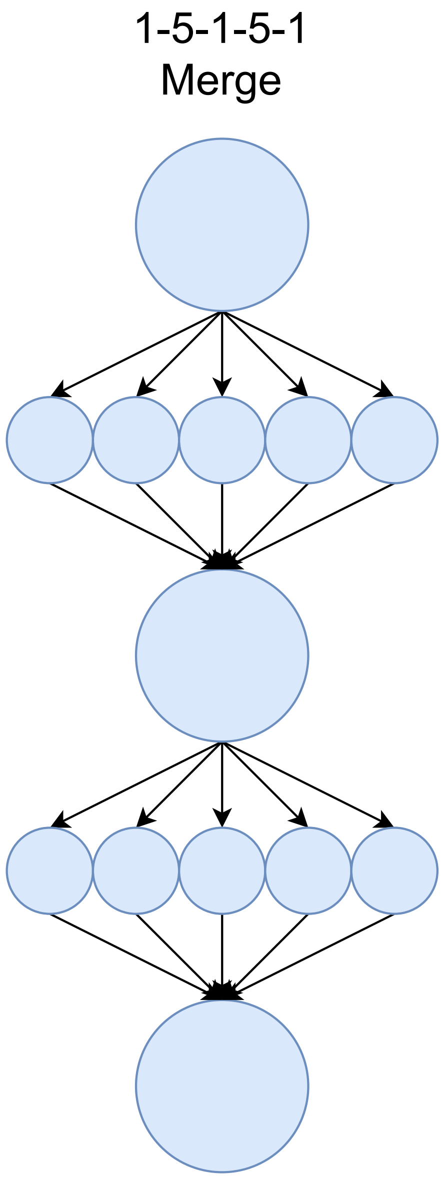

Figure A.3: Structure of 1-5-1-5-1 Join synthetic workflow.

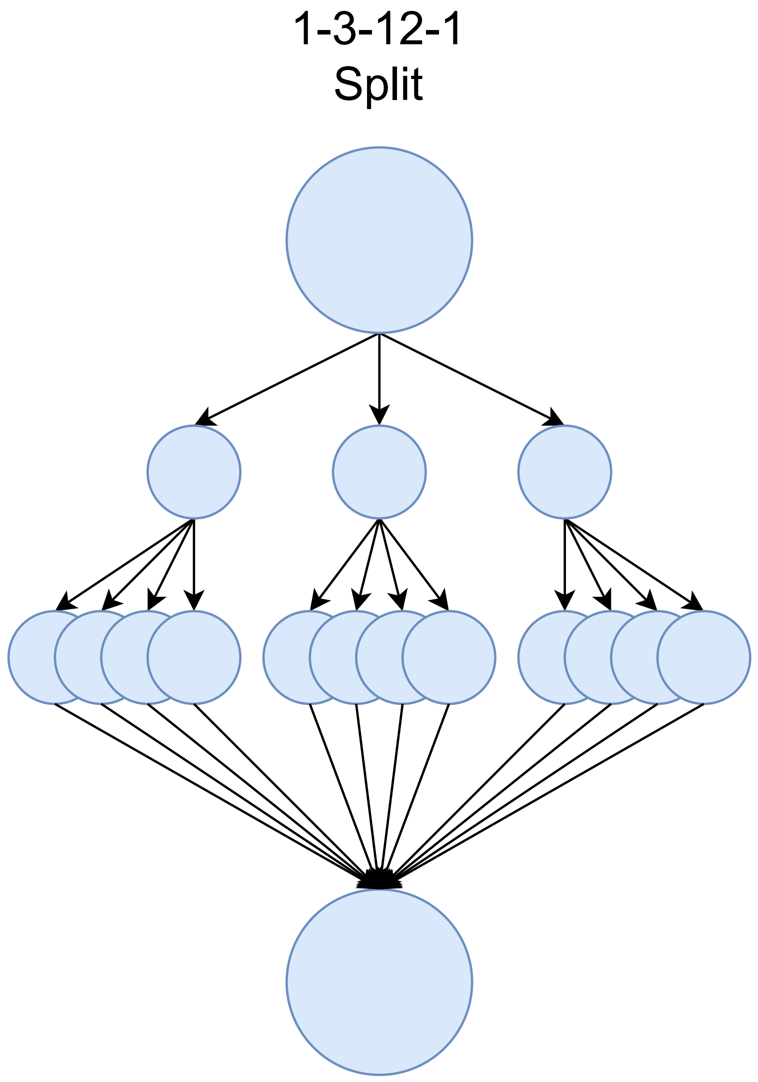

Figure A.4: Structure of 1-3-12-1 Split/Join synthetic workflow.

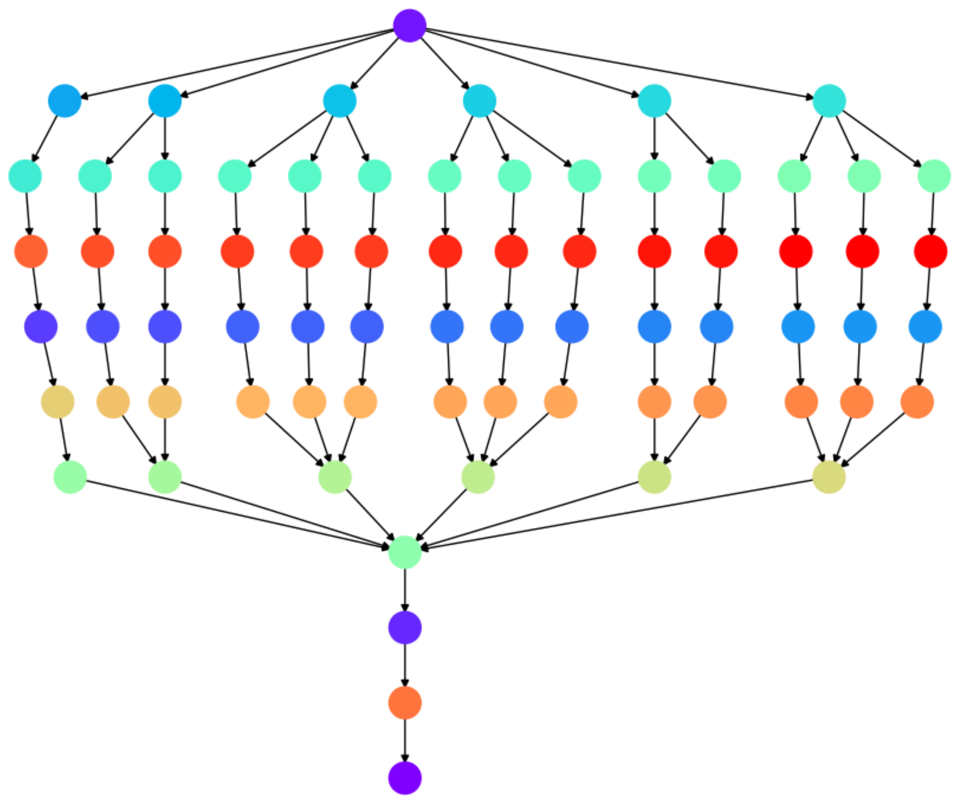

Figure A.5: Structure of Epigenomics workflow.

APPENDIX B
APPENDIX FOR SECTION 4.4

Table B.1: Calibrated Parameter Values for Single-Task. (disk bw: Disk Bandwidth; link bw: Link Bandwidth; pre: Pre-Execution Overhead; post: Post-Execution Overhead; flop: Flop rate).

Calibration Method	disk bw (MB/sec)	link bw (MB/sec)	pre (sec)	post (sec)	flop (flop/sec)
Manual	115	2500	180	180	1150000000
Exhaustive	1	200	1	400	1350000000
Random	36	683	82	612	2938073693
Hybrid	316	144	34	620	2373570891
Bayesian	330	122	341	337	2695289788

Table B.2: Calibrated Parameter Values for 1-1-1. (disk bw: Disk Bandwidth; link bw: Link Bandwidth; pre: Pre-Execution Overhead; post: Post-Execution Overhead; flop: Flop rate).

Calibration Method	disk bw (MB/sec)	link bw (MB/sec)	pre (sec)	post (sec)	flop (flop/sec)
Manual	115	2500	180	180	1150000000
Exhaustive	1	200	400	400	1350000000
Random	374	605	1761	38	1792763135
Hybrid	136	716	1585	192	2709639136
Bayesian	177	676	647	470	2247091827

Table B.3: Calibrated Parameter Values for Diamond. (disk bw: Disk Bandwidth; link bw: Link Bandwidth; pre: Pre-Execution Overhead; post: Post-Execution Overhead; flop: Flop rate).

Calibration Method	disk bw (MB/sec)	link bw (MB/sec)	pre (sec)	post (sec)	flop (flop/sec)
Manual	115	2500	180	180	1150000000
Exhaustive	1	200	800	400	800000000
Random	224	869	1016	248	765155150
Hybrid	186	757	1178	737	1840141677
Bayesian	54	850	1353	399	1134036926

Table B.4: Calibrated Parameter Values for Harpoon Join. (disk bw: Disk Bandwidth; link bw: Link Bandwidth; pre: Pre-Execution Overhead; post: Post-Execution Overhead; flop: Flop rate).

Calibration Method	disk bw (MB/sec)	link bw (MB/sec)	pre (sec)	post (sec)	flop (flop/sec)
Manual	115	2500	180	180	1150000000
Exhaustive	1	200	400	1600	1350000000
Random	56	37	191	1354	2880769401
Hybrid	80	754	1840	510	995301676
Bayesian	321	994	1372	1661	2610525414

Table B.5: Calibrated Parameter Values for 1-10-1. (disk bw: Disk Bandwidth; link bw: Link Bandwidth; pre: Pre-Execution Overhead; post: Post-Execution Overhead; flop: Flop rate).

Calibration Method	disk bw (MB/sec)	link bw (MB/sec)	pre (sec)	post (sec)	flop (flop/sec)
Manual	115	2500	180	180	1150000000
Exhaustive	1	200	1600	2000	800000000
Random	70	436	345	1658	562149900
Hybrid	279	316	1374	1657	676203054
Bayesian	361	24	1291	1558	2219058817

Table B.6: Calibrated Parameter Values for 1-5-1-5-1. (disk bw: Disk Bandwidth; link bw: Link Bandwidth; pre: Pre-Execution Overhead; post: Post-Execution Overhead; flop: Flop rate).

Calibration Method	disk bw (MB/sec)	link bw (MB/sec)	pre (sec)	post (sec)	flop (flop/sec)
Manual	115	2500	180	180	1150000000
Exhaustive	1	800	1600	2000	1900000000
Random	135	759	1682	1357	1082113114
Hybrid	193	272	1173	1333	950882204
Bayesian	157	333	936	1779	1241566609

Table B.7: Calibrated Parameter Values for 1-3-(4*)-1. (disk bw: Disk Bandwidth; link bw: Link Bandwidth; pre: Pre-Execution Overhead; post: Post-Execution Overhead; flop: Flop rate).

Calibration Method	disk bw (MB/sec)	link bw (MB/sec)	pre (sec)	post (sec)	flop (flop/sec)
Manual	115	2500	180	180	1150000000
Exhaustive	1	200	2000	2000	800000000
Random	24	29	343	1291	753558637
Hybrid	116	517	1532	1507	550531686
Bayesian	211	115	696	1726	526613252

Table B.8: Calibrated Parameter Values for Epigenomics. (disk bw: Disk Bandwidth; link bw: Link Bandwidth; pre: Pre-Execution Overhead; post: Post-Execution Overhead; flop: Flop rate).

Calibration Method	disk bw (MB/sec)	link bw (MB/sec)	pre (sec)	post (sec)	flop (flop/sec)
Manual	115	2500	180	180	1150000000
Exhaustive	1	1	800	400	2450000000
Random	127	915	1174	1169	632605578
Hybrid	293	607	1850	1951	1250586738
Bayesian	393	868	808	1229	612025322

Table B.9: Calibrated Parameter Values for Montage. (disk bw: Disk Bandwidth; link bw: Link Bandwidth; pre: Pre-Execution Overhead; post: Post-Execution Overhead; flop: Flop rate).

Calibration Method	disk bw (MB/sec)	link bw (MB/sec)	pre (sec)	post (sec)	flop (flop/sec)
Manual	115	2500	180	180	1150000000
Exhaustive	1	1	1600	1	2450000000
Random	335	62	991	40	578458231
Hybrid	44	501	1111	1586	589209254
Bayesian	389	28	872	739	578931049

Table B.10: Percent simulation error when using Exhaustive search with the ME metric.

Exhaustive / ME	Single Task	1-1-1	Diamond	Harpoon Join	1-10-1	1-5-1-5-1	1-3-(*4)-1	Epigenomics	Montage
Single Task		12.02%	36.45%	37.91%	52.82%	42.93%	53.55%	61.48%	61.08%
1-1-1			27.13%	32.56%	49.57%	39.74%	51.21%	57.84%	56.81%
Diamond				6.73%	31.83%	18.24%	33.93%	36.16%	26.63%
Harpoon Join					30.06%	10.99%	37.13%	24.95%	56.81%
1-10-1						26.47%	10.46%	15.00%	18.06%
1-5-1-5-1							31.69%	10.64%	54.89%
1-3-(*4)-1								18.66%	13.78%
Epigenomics									8.79%
Montage									

Table B.11: Percent simulation error when using Random search with the ME metric.

Random / ME	Single Task	1-1-1	Diamond	Harpoon Join	1-10-1	1-5-1-5-1	1-3-(*4)-1	Epigenomics	Montage
Single Task		8.53%	38.69%	47.25%	60.11%	51.59%	62.34%	69.14%	80.60%
1-1-1			18.67%	31.38%	49.61%	44.12%	52.85%	61.83%	51.56%
Diamond				5.63%	30.93%	18.13%	32.78%	36.36%	21.46%
Harpoon Join					33.16%	11.40%	38.40%	46.70%	78.27%
1-10-1						27.25%	6.32%	12.86%	4.64%
1-5-1-5-1							27.08%	13.43%	33.79%
1-3-(*4)-1								11.81%	26.33%
Epigenomics									5.85%
Montage									

Table B.12: Percent simulation error when using Hybrid search with the ME metric.

Hybrid / ME	Single Task	1-1-1	Diamond	Harpoon Join	1-10-1	1-5-1-5-1	1-3-(*4)-1	Epigenomics	Montage
Single Task		5.57%	36.59%	44.41%	58.06%	48.87%	60.15%	66.44%	76.95%
1-1-1			22.20%	37.21%	53.91%	48.62%	57.61%	65.91%	63.05%
Diamond				21.04%	43.38%	32.55%	48.48%	48.51%	58.54%
Harpoon Join					28.57%	16.13%	33.73%	32.35%	28.00%
1-10-1						24.66%	9.09%	11.62%	8.87%
1-5-1-5-1							26.61%	14.21%	32.73%
1-3-(*4)-1								20.90%	9.98%
Epigenomics									38.32%
Montage									

Table B.13: Percent simulation error when using Bayesian search with the ME metric.

Bayesian / ME	Single Task	1-1-1	Diamond	Harpoon Join	1-10-1	1-5-1-5-1	1-3-(*4)-1	Epigenomics	Montage
Single Task		24.16%	44.42%	49.90%	61.60%	55.00%	63.14%	73.17%	76.20%
1-1-1			28.46%	39.27%	54.85%	46.72%	57.62%	64.16%	69.23%
Diamond				14.15%	37.83%	27.09%	41.50%	44.19%	39.46%
Harpoon Join					31.35%	14.63%	41.21%	27.10%	64.63%
1-10-1						28.77%	7.92%	26.42%	60.32%
1-5-1-5-1							29.99%	12.87%	47.79%
1-3-(*4)-1								22.25%	5.30%
Epigenomics									7.07%
Montage									

Table B.14: Percent simulation error when using Exhaustive search with the TCMSE metric.

Exhaustive / TCMSE	Single Task	1-1-1	Diamond	Harpoon Join	1-10-1	1-5-1-5-1	1-3-(*4)-1	Epigenomics	Montage
Single Task		36.07%	45.77%	43.25%	56.06%	49.30%	55.89%	68.77%	56.81%
1-1-1			27.13%	32.56%	49.57%	39.74%	51.21%	57.84%	56.81%
Diamond				1.37%	28.58%	11.86%	31.58%	28.85%	30.91%
Harpoon Join					28.58%	11.87%	31.58%	28.87%	30.93%
1-10-1						23.28%	12.80%	11.35%	22.34%
1-5-1-5-1							22.19%	6.94%	30.93%
1-3-(*4)-1								18.66%	13.78%
Epigenomics									0.42%
Montage									

Table B.15: Percent simulation error when using Random search with the TCMSE metric.

Random / TCMSE	Single Task	1-1-1	Diamond	Harpoon Join	1-10-1	1-5-1-5-1	1-3-(*4)-1	Epigenomics	Montage
Single Task		38.99%	3.93%	10.83%	34.68%	19.37%	36.78%	36.92%	38.03%
1-1-1			22.43%	31.07%	48.91%	38.69%	51.27%	56.29%	59.11%
Diamond				7.18%	25.14%	4.25%	24.71%	39.73%	28.87%
Harpoon Join					26.76%	8.52%	26.66%	30.46%	16.76%
1-10-1						36.24%	1.13%	22.45%	1.99%
1-5-1-5-1							19.56%	9.05%	19.52%
1-3-(*4)-1								1.51%	23.62%
Epigenomics									3.27%
Montage									

Table B.16: Percent simulation error when using Hybrid search with the TCMSE metric.

Hybrid / TCMSE	Single Task	1-1-1	Diamond	Harpoon Join	1-10-1	1-5-1-5-1	1-3-(*4)-1	Epigenomics	Montage
Single Task		23.80%	41.94%	44.57%	57.58%	50.23%	58.52%	68.87%	66.35%
1-1-1			26.71%	35.72%	52.18%	46.57%	54.58%	64.66%	54.63%
Diamond				28.07%	47.69%	40.27%	51.67%	57.30%	55.14%
Harpoon Join					33.33%	16.71%	41.50%	33.77%	60.44%
1-10-1						25.43%	6.83%	11.65%	0.49%
1-5-1-5-1							28.02%	4.11%	56.46%
1-3-(*4)-1								18.63%	11.25%
Epigenomics									15.17%
Montage									

Table B.17: Percent simulation error when using Bayesian search with the TCMSE metric.

Bayesian / TCMSE	Single Task	1-1-1	Diamond	Harpoon Join	1-10-1	1-5-1-5-1	1-3-(*4)-1	Epigenomics	Montage
Single Task		12.44%	36.21%	38.90%	53.66%	44.38%	54.65%	62.84%	62.43%
1-1-1			30.96%	41.24%	56.16%	47.44%	58.80%	64.82%	73.37%
Diamond				21.02%	43.37%	33.15%	48.46%	49.20%	56.80%
Harpoon Join					29.25%	9.22%	35.34%	23.72%	53.63%
1-10-1						31.89%	7.06%	29.06%	73.73%
1-5-1-5-1							20.96%	16.74%	7.94%
1-3-(*4)-1								22.48%	4.10%
Epigenomics									37.88%
Montage									

BIBLIOGRAPHY

[1] OPNET modeler and ns-2: comparing the accuracy of network simulators for packet-level analysis using a network testbed. *WSEAS Transactions on Computers*, 2:700–707, 2003.

[2] Laksono Adhianto, Sinchan Banerjee, Mike Fagan, Mark Krentel, Gabriel Marin, John Mellor-Crummey, and Nathan R Tallent. HPCToolkit: Tools for performance analysis of optimized parallel programs. *Concurrency and Computation: Practice and Experience*, 22(6):685–701, 2010.

[3] Malcolm Atkinson, Sandra Gesing, Johan Montagnat, and Ian Taylor. Scientific workflows: Past, present and future. *Future Generation Computer Systems*, 75:216–227, 2017.

[4] The GPyOpt authors. GPyOpt: A bayesian optimization framework in python. `http://github.com/SheffieldML/GPyOpt`, 2016.

[5] Rajive Bagrodia, Ewa Deelman, and Thomas Phan. Parallel Simulation of Large-Scale Parallel Applications. *IJHPCA*, 15(1):3–12, 2001.

[6] Emir M Bahsi, Emrah Ceyhan, and Tevfik Kosar. Conditional workflow management: A survey and analysis. *Scientific Programming*, 15(4):283–297, 2007.

[7] Adam Barker and Jano Van Hemert. Scientific workflow: a survey and research directions. In *Parallel Processing and Applied Mathematics*, pages 746–753. Springer, 2007.

[8] Ingmar Baumgart, Bernhard Heep, and Stephan Krause. OverSim: A Flexible Overlay Network Simulation Framework. In *Proc. of the 10th IEEE Global Internet Symposium*, pages 79–84. IEEE, May 2007.

[9] P. Bedaride, A. Degomme, S. Genaud, A. Legrand, G. Markomanolis, M. Quinson, M. Stillwell, F. Suter, and B. Videau. Toward Better Simulation of MPI Applications on Ethernet/TCP Networks. In *Prod. of the 4th Intl. Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*, 2013.

[10] William H. Bell, David G. Cameron, A. Paul Millar, Luigi Capozza, Kurt Stockinger, and Floriano Zini. OptorSim - A Grid Simulator for Studying Dynamic Data Replication Strategies. *IJHPCA*, 17(4):403–416, 2003.

[11] Marc Bux and Ulf Leser. Parallelization in scientific workflow management systems. *arXiv preprint arXiv:1303.7195*, 2013.

[12] Rajkumar Buyya and Manzur Murshed. GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. *Concurrency and Computation: Practice and Experience*, 14(13-15):1175–1220, December 2002.

[13] Hollowell C., Barnett J., Caramarcu C., Strecker-Kellogg W., Wong A., and Zaytsev A. Mixing HTC and HPC Workloads with HTCondor and Slurm. *Journal of Physics: Conference Series*, 2017.

[14] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, Cesar A. F. De Rose, and Rajkumar Buyya. CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms. *Software: Practice and Experience*, 41(1):23–50, January 2011.

[15] C. D. Carothers, D. Bauer, and S. Pearce. ROSS: A High-Performance, Low Memory, Modular Time Warp System. In *Proc. of the 14th ACM/IEEE/SCS Workshop of Parallel on Distributed Simulation*, pages 53–60, 2000.

[16] H. Casanova, A. Giersch, A. Legrand, M. Qinson, and F. Suter. Versatile, Scalable, and Accurate Simulation of Distributed Applications and Platforms. *Journal of Parallel and Distributed Computing*, 75(10):2899–2917, 2014.

[17] Henri Casanova, Rafael Ferreira da Silva, Ryan Tanaka, Suraj Pandey, Gautam Jethwani, William Koch, Spencer Albrecht, James Oeth, and Frédéric Suter. Developing accurate and scalable simulators of production workflow management systems with wrench. *Future Generation Computer Systems*, 112:162–175, 2020.

[18] Henri Casanova, Arnaud Giersch, Arnaud Legrand, Martin Quinson, and Frédéric Suter. Versatile, scalable, and accurate simulation of distributed applications and platforms. *Journal of Parallel and Distributed Computing*, 74(10):2899 – 2917, 2014.

[19] Henri Casanova, Suraj Pandey, James Oeth, Ryan Tanaka, Frederic Suter, and Rafael Ferreira da Silva. WRENCH: A Framework for Simulating Workflow Management Systems. In *13th Workshop on Workflows in Support of Large-Scale Science (WORKS'18)*, pages 74–85, 2018.

[20] Chameleon. `https://www.chameleoncloud.org`, 2020.

[21] XENON Collaboration, XENON100 Collaboration, DARWIN Collaboration, E Aprile, et al. Observation of two-neutrino double electron capture in 124xe with xenon1t. *Nature*, 568:532–535, 2019.

[22] Ewa Deelman, Dennis Gannon, Matthew Shields, and Ian Taylor. Workflows and e-science: An overview of workflow system features and capabilities. *Future Generation Computer Systems*, 25(5):528–540, 2009.

[23] Ewa Deelman, Tom Peterka, Ilkay Altintas, Christopher D Carothers, Kerstin Kleese van Dam, Kenneth Moreland, Manish Parashar, Lavanya Ramakrishnan, Michela Taufer, and Jeffrey Vetter. The future of scientific workflows. *The International Journal of High Performance Computing Applications*, 32(1):159–175, 2018.

[24] Ewa Deelman, Karan Vahi, Gideon Juve, Mats Rynge, Scott Callaghan, Philip J Maechling, Rajiv Mayani, Weiwei Chen, Rafael Ferreira da Silva, Miron Livny, and Kent Wenger. Pegasus: a workflow management system for science automation. *Future Generation Computer Systems*, 46:17–35, 2015.

[25] Ewa Deelman, Karan Vahi, Gideon Juve, Mats Rynge, Scott Callaghan, Philip J Maechling, Rajiv Mayani, Weiwei Chen, Rafael Ferreira da Silva, Miron Livny, and Kent Wenger. Pegasus: a Workflow Management System for Science Automation. *Future Generation Computer Systems*, 46:17–35, 2015.

[26] Ewa Deelman, Karan Vahi, Mats Rynge, Rajiv Mayani, Rafael Ferreira da Silva, George Papadimitriou, and Miron Livny. The evolution of the pegasus workflow management software. *Computing in Science Engineering*, 21(4):22–36, 2019.

[27] A. Degomme, A. Legrand, G. Markomanolis, M. Quinson, M. Stillwell, and F. Suter. Simulating MPI applications: the SMPI approach. *IEEE Transactions on Parallel and Distributed Systems*, 28:2387–2400, 2017.

[28] Rafael Ferreira da Silva, Rosa Filgueira, Ilia Pietri, Ming Jiang, Rizos Sakellariou, and Ewa Deelman. A characterization of workflow management systems for extreme-scale applications. *Future Generation Computer Systems*, 75:228–238, 2017.

[29] Rafael Ferreira da Silva, Loïc Pottier, Tainã Coleman, Ewa Deelman, and Henri Casanova. Workflowhub: Community framework for enabling scientific workflow research and development. In *15th Workshop on Workflows in Support of Large-Scale Science (WORKS'20)*, 2020.

[30] James Frey. Condor dagman: Handling inter-job dependencies. *University of Wisconsin, Dept. of Computer Science, Tech. Rep*, 2002.

[31] K. Fujiwara and H. Casanova. Speed and Accuracy of Network Simulation in the SimGrid Framework. In *Proc. of the 1st Intl. Workshop on Network Simulation Tools*, 2007.

[32] P. Pablo Garrido, Manuel P. Malumbres, and Carlos T. Calafate. Ns-2 vs. OPNET: A Comparative Study of the IEEE 802.11e Technology on MANET Environments. In *Proceedings*

of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops, Simutools '08, Brussels, BEL, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[33] F. C. Heinrich, T. Cornebize, A. Degomme, A. Legrand, A. Carpen-Amarie, S. Hunold, A. Orgerie, and M. Quinson. Predicting the energy-consumption of mpi applications at scale using only a single node. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 92–102, 2017.

[34] Torsten Hoefler, Timo Schneider, and Andrew Lumsdaine. LogGOPSim - Simulating Large-Scale Applications in the LogGOPS Model. In *Proc. of the ACM Workshop on Large-Scale System and Application Performance*, pages 597–604, June 2010.

[35] John Hourdakis, Panos G Michalopoulos, and Jiji Kottommannil. Practical procedure for calibrating microscopic traffic simulation models. *Transportation research record*, 1852(1):130–139, 2003.

[36] Philipp Hurni and Torsten Braun. Calibrating Wireless Sensor Network Simulation Models with Real-World Experiments. In Luigi Fratta, Henning Schulzrinne, Yutaka Takahashi, and Otto Spaniol, editors, *NETWORKING 2009, 8th International IFIP-TC 6 Networking Conference, Aachen, Germany, May 11-15, 2009. Proceedings*, volume 5550 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2009.

[37] Kate Keahey, Jason Anderson, Zhuo Zhen, Pierre Riteau, Paul Ruth, Dan Stanzione, Mert Cevik, Jacob Colleran, Haryadi S. Gunawi, Cody Hammock, Joe Mambretti, Alexander Barnes, François Halbach, Alex Rocha, and Joe Stubbs. Lessons Learned from the Chameleon Testbed. In *Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC '20)*. USENIX Association, July 2020.

[38] G. Kecskemeti. DISSECT-CF: A simulator to foster energy-aware scheduling in infrastructure clouds. *Simulation Modelling Practice and Theory*, 58(2):188–218, 2015.

[39] Alexei Klimentov, P Buncic, K De, Shantenu Jha, T Maeno, R Mount, P Nilsson, D Oleynik, S Panitkin, A Petrosyan, et al. Next generation workload management system for big data on heterogeneous distributed computing. In *Journal of Physics: Conference Series*, volume 608, page 012040. IOP Publishing, 2015.

[40] A. Lèbre, A. Legrand, F. Suter, and P. Veyre. Adding Storage Simulation Capacities to the SimGrid Toolkit: Concepts, Models, and API. In *Proc. of the 8th IEEE Intl. Symp. on Cluster Computing and the Grid*, 2015.

[41] A. Lebre, A. Legrand, F. Suter, and P. Veyre. Adding Storage Simulation Capacities to the SimGrid Toolkit: Concepts, Models, and API. In *Proc. 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 251–260, 2015.

[42] J. Lessmann, P. Janacik, L. Lachev, and D. Orfanus. Comparative Study of Wireless Network Simulators. In *Seventh International Conference on Networking (icn 2008)*, pages 517–523, 2008.

[43] Chee Sun Liew, Malcolm P Atkinson, Michelle Galea, Tan Fong Ang, Paul Martin, and Jano I Van Hemert. Scientific workflows: Moving across paradigms. *ACM Computing Surveys (CSUR)*, 49(4):66, 2017.

[44] S. Lim, B. Sharma, G. Nam, E. K. Kim, and C. R. Das. MDCSim: A multi-tier data center simulation, platform. In *Proc. IEEE International Conf. on Cluster Computing and Workshops*, pages 1–9, 2009.

[45] Ji Liu, Esther Pacitti, Patrick Valduriez, and Marta Mattoso. A survey of data-intensive scientific workflow management. *Journal of Grid Computing*, 13(4):457–493, 2015.

[46] Y.B. Liu, Okke Batelaan, F. Smedt, Jana Poorova, and L. Velcicka. Automated calibration applied to a GIS-based flood simulation model using PEST. *Floods, from Defence to Management*, pages 317–326, 01 2005.

[47] A. W. Malik, K. Bilal, K. Aziz, D. Kliazovich, N. Ghani, S. U. Khan, and R. Buyya. Cloudnetsim++: A toolkit for data center simulations in omnet++. In *2014 11th Annual High Capacity Optical Networks and Emerging/Enabling Technologies (Photonics for Energy)*, pages 104–108, 2014.

[48] Alberto Montresor and Márk Jelasity. PeerSim: A Scalable P2P Simulator. In *Proc. of the 9th Intl. Conf. on Peer-to-Peer*, pages 99–100, September 2009.

[49] A. W. Mu'alem and D. G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *IEEE Transactions on Parallel and Distributed Systems*, 12(6):529–543, 2001.

[50] A. Núñez, J. Fernández, J. Carretero, J. D. García, and L. Prada. SIMCAN: A Simulator Framework for Computer Architectures and Storage Networks. In *Proc. of the 1st International Workshop on OMNeT++*, 2008.

[51] A. Núñez, J. Vázquez-Poletti, A. Caminero, J. Carretero, and I. M. Llorente. Design of a New Cloud Computing Simulation Platform. In *Proc. of the 11th Intl. Conf. on Computational Science and its Applications*, pages 582–593, June 2011.

[52] Simon Ostermann, Radu Prodan, and Thomas Fahringer. Dynamic Cloud Provisioning for Scientific Grid Workflows. In *Proc. of the 11th ACM/IEEE Intl. Conf. on Grid Computing (Grid)*, pages 97–104, 2010.

[53] M. Pelikan. *Hierarchical Bayesian Optimization Algorithm.* Springer Verlag, 2005.

[54] Martin Pelikan, David E. Goldberg, and Erick Cantú-Paz. BOA: The Bayesian Optimization Algorithm. In *Proc. of the 1st Annual Conference on Genetic and Evolutionary Computation*, pages 525–532, 1999.

[55] Laurent Pouilloux, Takahiro Hirofuchi, and Adrien Lebre. SimGrid VM: Virtual Machine Support for a Simulation Framework of Distributed Systems. *IEEE transactions on cloud computing*, September 2015.

[56] T. Qayyum, A. W. Malik, M. A. Khan Khattak, O. Khalid, and S. U. Khan. FogNetSim++: A Toolkit for Modeling and Simulation of Distributed Fog Environment. *IEEE Access*, 6:63570–63583, 2018.

[57] A. S. M. Rizvi, T. R. Toha, M. M. R. Lunar, M. A. Adnan, and A. B. M. A. A. Islam. Cooling energy integration in simgrid. In *2017 International Conference on Networking, Systems and Security (NSysS)*, pages 132–137, 2017.

[58] The SimGrid Project. `https://simgrid.org/`, 2020.

[59] L. Stanisic. *A Reproducible Research Methodology for Designing and Conducting Faithful Simulations of Dynamic HPC Applications.* PhD thesis, Université Grenoble Alpes, France, 2015.

[60] L. Stanisic, E. Agullo, A. Buttari, A. Guermouche, A. Legrand, F. Lopez, and B. Videau. Fast and accurate simulation of multithreaded sparse linear algebra solvers. In *2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS)*, pages 481–490, 2015.

[61] Ian J Taylor, Ewa Deelman, Dennis B Gannon, Matthew Shields, et al. *Workflows for e-Science: scientific workflows for grids*, volume 1. Springer, 2007.

[62] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice: the Condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356, 2005.

[63] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice: the Condor experience. *Concurrency and computation: practice and experience*, 17(2-4):323–356, 2005.

[64] Mustafa Tikir, Michael Laurenzano, Laura Carrington, and Allan Snavely. PSINS: An Open Source Event Tracer and Execution Simulator for MPI Applications. In *Proc. of the 15th Intl. Euro-Par Conf. on Parallel Processing*, number 5704 in LNCS, pages 135–148. Springer, August 2009.

[65] T.R. T.R. Andel and A. Yasinsac. On the credibility of MANET simulations. *Computer*, 39(7):48–54, 2006.

[66] P. Velho and A. Legrand. Accuracy Study and Improvement of Network Simulation in the SimGrid Framework. In *Proc. of the 2nd Intl. Conf. on Simulation Tools and Techniques*, 2009.

[67] P. Velho, L. Mello Schnorr, H. Casanova, and A. Legrand. On the Validity of Flow-level TCP Network Models for Grid and Cloud Simulations. *ACM Transactions on Modeling and Computer Simulation*, 23(4), 2013.

[68] Elizabeth G. Wilbanks, David J. Larsen, Russell Y. Neches, Andrew I. Yao, Chia-Ying Wu, Rachel A. S. Kjolby, and Marc T. Facciotti. A workflow for genome-wide mapping of archaeal transcription factors with ChIP-seq. *Nucleic Acids Research*, 40(10):e74–e74, 02 2012.

[69] The WRENCH Project. `http://wrench-project.org`, 2018.

[70] Tao Yang, Yiqun Pan, Jiachen Mao, Yonglong Wang, and Zhizhong Huang. An automated optimization method for calibrating building energy simulation models with measured data: Orientation and a case study. *Applied Energy*, 179:1220 – 1231, 2016.

[71] A. B. Yoo, M. A. Jette, and M. Grondona. SLURM: Simple linux utility for resource management. In *Proc. 9th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, pages 44–60, June 2003.

[72] Jia Yu and Rajkumar Buyya. A taxonomy of workflow management systems for grid computing. *Journal of Grid Computing*, 3(3-4):171–200, 2005.

[73] Gengbin Zheng, Gunavardhan Kakulapati, and Laxmikant Kalé. BigSim: A Parallel Simulator for Performance Prediction of Extremely Large Parallel Machines. In *Proc. of the 18th Intl. Parallel and Distributed Processing Symposium (IPDPS)*, April 2004.

[74] Lina Zhou, Shimei Pan, Jianwu Wang, and Athanasios V Vasilakos. Machine learning on big data: Opportunities and challenges. *Neurocomputing*, 237:350–361, 2017.