Determining Levels of Detail for Simulators of Parallel and Distributed Computing Systems via Automated Calibration

Jesse McDonald
Information and Computer Sciences
University of Hawai'i at Mānoa
Honolulu, Hawai'i, USA

Yick-Ching Wong
Information and Computer Sciences
University of Hawai'i at Mānoa
Honolulu, Hawai'i, USA

Kshitij Mehta
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA

Frédéric Suter
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA

Rafael Ferreira da Silva
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA

Loïc Pottier
Lawrence Livermore National
Laboratory
Livermore, California, USA

Ewa Deelman
Information Sciences Institute
University of Southern California
Marina Del Rey, California, USA

Henri Casanova
Information and Computer Sciences
University of Hawai'i at Mānoa
Honolulu, Hawai'i, USA

Abstract

There are two sources of inaccuracy when simulating parallel and distributed computing systems: (i) a simulator implemented at an insufficient level of detail; and (ii) incorrectly calibrated simulation parameter values. Increasing the simulator's level of detail can improve accuracy, but at the cost of higher space, time, and/or software complexity. Furthermore, evaluating the intrinsic accuracy of a simulator requires that its parameters be well-calibrated. Making decisions regarding the level of detail is thus challenging. We propose a methodology for instantiating the simulation calibration process and a framework for automating this process, which makes it possible to pick appropriate levels of detail for any simulator. We demonstrate the usefulness of our approach via two case studies for two different domains.

CCS Concepts

• **Computing methodologies** → **Modeling and simulation**; **Simulation tools**; **Parallel computing methodologies**; **Distributed computing methodologies**.

Keywords

Simulation of parallel and distributed computing systems, simulation accuracy, simulation calibration

1 Introduction

Parallel and Distributed Computing (PDC) research often involves executing application workloads on hardware platforms. Many researchers resort to simulating these executions because simulation makes it possible to explore hypothetical scenarios, can yield 100% reproducible results, and can require less time, labor, carbon footprint, and/or funding. The main concern with simulation is *accuracy*, i.e., how representative simulated executions are of ground-truth, real-world executions. A common way to improve simulation accuracy is to increase the level of detail at which real-world behaviors are simulated. But doing so incurs costs (higher space, time, and/or software complexity), raising the question: at which level of detail should a simulator be implemented? [43]

Consider a PDC system of interest and a simulator of that system. Key questions are whether the simulator can achieve some desired accuracy at its current level of detail, whether a higher level of detail is required, or whether a lower lever of detail is tolerable. Answering these questions is challenging because simulation error not only comes from the implemented level of detail, but also from possibly incorrect values of the user-selected parameters that define the behavior of the simulation models. To evaluate the intrinsic accuracy of a simulator soundly, these values must themselves be accurate. Unfortunately, selecting accurate values, or *calibrating* [47] the parameters, is a non-trivial optimization problem with dimensionality in the number of parameters. Thus, there is a tension that further deepens the above challenge: increasing the level of detail can improve simulation accuracy, but often introduces more parameters, which makes simulation calibration more difficult [44].

To address the above challenge we propose to use automated simulation calibration, making the following contributions:

- A general, automated simulation calibration framework;

- A methodology for instantiating this framework to evaluate the intrinsic accuracy of a simulator;
- A demonstration of the usefulness of this methodology for two domains in which simulation is used routinely: (i) scientific workflows; and (ii) message-passing applications.

The rest of the paper is organized as follows. Section 2 discusses related work. Sections 3 and 4 detail our approach, which we evaluate via two case studies in Sections 5 and 6. Section 7 concludes with a summary of results and future work.

2 Background and Related Work

Many frameworks are available for developing PDC simulators. Some implement high levels of detail, often using Parallel Discrete Event Simulation [24] to increase simulation scalability [7, 10, 38, 46], while others implement lower levels of detail [6, 8, 12, 32, 35, 39, 48, 52]. The choice of a particular simulation framework impacts the level of detail of the base simulation abstractions and models. But a simulator can use these abstractions and models in arbitrary ways. For instance, although a simulation framework may provide ways to simulate the network topology at a high level of detail, a simulator could opt for a low level of detail by abstracting away the entire network as a single shared "macro" network link. In general, a simulator can implement different levels of detail for simulating different components of a system, somewhat independently of the simulation framework used. Picking an appropriate level of detail for a simulator is challenging in general [5, 15] and thus also for simulators of PDC systems [1, 9, 20, 41, 42, 49]. In this work, we tackle this challenge via automated simulation calibration.

The need to calibrate model parameters with respect to ground-truth experimental data has long been recognized [47]. Calibration may seem unnecessary when simulating computer systems since simulation parameter values should come directly from hardware/software specifications. Previous research shows this not to be the case, even when simulating at high levels of detail [23, 25, 29, 34]. The lower the level of detail, the more abstract the simulation models and their parameters, whose correct values are then complex functions of the hardware/software specifications. To the best of our knowledge there is no standard calibration method for simulators of PDC systems. In [36], the authors review 114 research publications that include results obtained with a particular PDC simulation framework, and find that calibration is typically not performed nor documented. When documented, authors describe labor-intensive, use case-specific, and (at least partially) manual procedures. In this work we develop and use an automated simulation calibration framework. Automated calibration has been explored recently for two specific PDC use cases [36, 37], using some of the classical optimization algorithms we also consider in this work. The goal in these previous works is to improve the accuracy of a simulator implemented at an already decided level of detail, and to show improvement over manual calibration done by human experts. By contrast, this work uses automated calibration as a means to address the challenge of picking an appropriate level of detail.

3 Proposed Methodology

Consider a PDC system of interest, i.e., a class of application workloads to execute on a class of platforms, for which ground-truth execution data is available. Our approach consists in automatically calibrating any simulator of this system, so that one can soundly evaluate its intrinsic accuracy, and decide whether to implement a new simulator at a different level of detail. This approach is predicated on automatically solving the simulation calibration optimization problem: Given a function that quantifies the discrepancy between ground-truth and simulated executions, which we term the *loss function*, and given a simulator whose behavior is defined by various parameters, find the parameter values that minimize the loss function. We propose a methodology for instantiating the automated simulation calibration process as follows.

Pick parameter ranges – The constraints of the optimization problem are user-specified ranges for the parameter values. The more knowledge the user possesses about the target system, the narrower these ranges and the more tractable the optimization problem. But the user should be able to pick overly broad ranges since knowledge about the target system is often incomplete.

Pick a loss function – A scalar metric is often used to quantify execution performance (e.g., the execution time), and a natural idea is to define loss functions based on such metrics (e.g., the average relative difference between real-world and simulated execution times). Other metrics that capture the temporal (when events happen) or physical (on which hardware resources events happen) structure of the execution may yield more robust calibrations. Such metrics reduce the possibility for simulation errors of different simulation models to compensate for each other (i.e., overestimating I/O performance while underestimating compute performance). Many loss functions can be defined, but selecting the most appropriate one is difficult because the values of different loss functions are not easily comparable. Even for the best possible calibration (i.e., the best set of simulation parameter values), different loss functions can have different (non-zero) values. Furthermore, when different loss functions lead to different calibrations one does not know which one is the closest to the (unknown) best calibration.

To resolve this difficulty we use a classical *synthetic benchmarking* technique: we pick arbitrary simulation parameter values and simulate executions for all application workloads and platform configurations in our ground-truth data. We obtain synthetic ground-truth data for which we know the best calibration by design. We then perform automated calibration using this data for each loss function, each leading to a particular calibration. We compute the relative L1 distance between each of these calibrations and the best calibration, and then pick the loss function that achieves the lowest distance, which we term the *calibration error*.

Pick an optimization algorithm – In principle, any optimization algorithm can be used to compute a calibration, but different algorithms may have different relative effectiveness for different use cases. For instance, for low-dimensionality scenarios with narrow user-specified parameter ranges, random search has been shown to be effective [36]. However, the choice of the algorithm is not necessarily orthogonal to the choice of the loss function. Our methodology thus consists in using the synthetic benchmarking technique described above for each combination of loss function and optimization algorithm, so as to select the loss/algorithm pair that achieves the lowest calibration error.

Pick a calibration time budget – Each loss function evaluation by the optimization algorithm entails invoking the simulator for

each ground-truth data point, which has non-zero computational cost. In practice, the user may wish to allocate a fixed time budget to the calibration process. We do so ourselves in this work to enable fair comparison of different calibration options.

4 Implementation

We have prototyped an automated simulation calibration framework as a Python package, making no assumptions regarding the, necessarily use case-specific, simulator. It provides a `Simulator` class with a `run()` method to be overridden for invoking the simulator. It returns a loss value computed by a user-provided loss function implementation. Our framework relies on the `multiprocessing` package for parallel calibration on multiple cores. Users can define continuous and discrete parameters to be calibrated within any arbitrary range, using the following iterative algorithms:

Grid search (GRID) – An algorithm that performs an exhaustive search over points in a discretized grid over the parameter space, doubling the resolution of the discretization at each iteration.

Random search (RAND) – An algorithm that samples a random point in the search space at each iteration.

Gradient descent (GRAD) – An algorithm that, at each iteration, randomly samples a point in the search space and performs a gradient descent using that point as a starting point until convergence.

Bayesian Optimization (BO) – An algorithm that uses an incrementally updated surrogate model for learning the relationship between the loss function's inputs and outputs. This model is used to prune the search space and identify promising regions, balancing exploration and exploitation. While the exploration samples input configurations that can potentially improve the accuracy of the surrogate model, the exploitation samples input configurations that are predicted by the model to be high-performing. We use the BO implementation in the `scikit-optimize` package [45], using four possible regressors: Gaussian Process (BO-GP), Random Forest (BO-RF), Extra Trees (BO-ET), or Gradient Boosting Quantile Regressor Trees (BO-GBRT).

Our framework allows the user to specify a bound on the elapsed time before a solution is returned, so that a fixed calibration time budget can be used regardless of the algorithm. In this work we omit results for the GRID and GRAD algorithms because they performed poorly in preliminary experiments. We also found that all versions of the BO algorithms perform almost identically, and we only present results for the BO-GP algorithm. The code of our calibration framework is available at [3].

5 Case Study #1: Scientific Workflows

Scientific workflows and their executions on PDC platforms have supported some of the most significant discoveries of the past decades [4, 17]. In this context, many researchers have used simulation to explore relevant questions (often to investigate scheduling and resource management strategies), which has motivated the development of simulation framework specifically for this purpose [2, 14, 51]. In this section we apply our approach to a simulator that was designed to evaluate workflow scheduling strategies. All relevant datasets and software are available at [3].

5.1 Ground Truth

The ground truth that was made available to us was produced using WfCommons tools [16] to generate and execute realistic workflow benchmarks that are based on the structures and the execution logs of real-world workflows. Benchmarks were generated for five different scientific applications, five different workflow sizes (i.e., numbers of tasks), five different amounts of per-task CPU work, and four different total data footprint sizes (i.e., sum of the sizes of all workflow data files). Additionally, benchmark were also generated for two synthetic workflow patterns, a "chain" linear task graph and a "forkjoin" fan-out/fan-in task graph, each for three different workflow sizes, five different amounts of per-task CPU work, and three different total data footprint sizes.

These benchmarks were executed with the Pegasus [18] (v5.0.3) / HTCondor [28] (v24.0) Workflow Management System (WMS) on Chameleon Cloud [13]. Pegasus and the HTCondor Central Manager were installed on a "submit node," and n HTCondor workers were deployed on n "worker nodes" (48-core Intel Icelake processors running Ubuntu 22.04) on which tasks execute. The workflow's input data was initially stored on disk at the submit node, and all workflow data was transferred (automatically by Pegasus) between the submit node and the worker nodes until all output data was eventually stored on the submit node. Each benchmark was executed five times for a deployment with $n = 1, 2, 4, 6$ workers (except for the chain benchmark, which only uses $n = 1$ worker). Some benchmark executions with high data footprint and small workflow sizes are not available due to limits imposed on individual file sizes. In total, execution logs were collected from 9,200 workflow executions (i.e., 1,840 different executions, each repeated five times). Table 1 gives more details on the ground-truth data.

5.2 Simulator Versions

Our target simulator was developed in C++ using the WRENCH simulation framework [11]. It takes as input a workflow specification, as a WfCommons JSON file, and a number of workers. In practice a user would decide to implement different levels of detail only as needed (as determined by using the methodology proposed in this work). For the purpose of this case study, however, we have modified the simulator so that it takes in three arguments for specifying the level of detail for simulating three particular components: (i) the network; (ii) the storage system; and (iii) the compute system.

We do not have precise information regarding the physical network topology that interconnects the submit node and the workers, besides the fact that all nodes are in the same data-center, and perhaps in the same rack. As a result, we consider three standard options for the network topology: (i) a single shared network link; (ii) a star topology of dedicated network links between the submit node and each worker; and (iii) a single shared network link out of the submit node and then a dedicated network link to each workers.

The storage system consists of disks attached to nodes and of services used for reading/writing workflow data on these disks. We consider two options: (i) only the submit node has storage capabilities; and (ii) the submit node and the workers all have storage capabilities. In the physical platform all nodes have storage capabilities, but it is possible that simulating storage only on the

Table 1: Workflow specifications used for ground-truth executions logs.

Workflow Application	Workflow Size (#tasks)[1]	Sequential Work / task (sec.)[2]	Data Footprint (MB)[3]
Epigenomics [50] (bioinformatics)	43, 64, 86, 129, 215	0.6, 1.15, 1.73, 7.22, 73.25	0, 150, 1500, 15000
1000Genome [50] (bioinformatics)	54, 81, 108, 162, 270	0.9, 1.47, 2.11, 8.02, 80.94	0, 150, 1500, 15000
SoyKB [50] (bioinformatics)	98, 147, 196, 294, 490	0.53, 1.06, 1.6, 6.55, 74.21	0, 150, 1500, 15000
Montage [50] (astronomy)	60, 90, 120, 180, 300	0.59, 1.12, 1.75, 7.07, 73.13	0, 150, 1500, 15000
Seismology [50] (seismology)	103, 154, 206, 309, 515	0.74, 1.28, 1.91, 8.34, 86.25	0, 150, 1500, 15000
Chain (synthetic)	10, 25, 50	0.83, 1.36, 1.85, 5.74, 48.94	0, 150, 1500
Forkjoin (synthetic)	10, 25, 50	0.84, 1.39, 2.05, 7.61, 70.76	0, 150, 1500

[1] the smallest workflow size is the smallest size that can be generated by WfCommons, and the other sizes are approximately 1.5x, 2x, 3x, and 5x larger (WfCommons enforces constraints on workflow size to ensure that the generated task-graphs are representative of the original application).
[2] based on an execution on a single core of a worker node.
[3] given as the sum of the sizes of all the data files used by the workflow, including intermediate files.

submit node is sufficient or even more representative, performance-wise, of what happens in practice.

The compute system consists of compute services on workers, accessed by the WMS to execute workflow tasks. We consider two options: (i) the WMS submits tasks directly to the workers; and (ii) the WMS uses HTCondor to access the workers. WRENCH provides built-in abstractions for simulating HTCondor deployments.

Table 2 depicts the above options and shows the parameters to calibrate, which lead to $2 \times 2 \times 3 = 12$ simulator versions. The highest level of detail for all three options leads to 10 parameters. The simulator could take many more parameters, e.g., a different disk bandwidth at each worker. Furthermore, the simulation models provided by the simulation framework also come with their own parameters. For instance, the size in bytes of each control message exchanged between simulated components is a parameter, which defaults to 0 bytes. Instead of considering possibly hundreds of parameters, and to ensure that our case study is representative of common practice in the field, we follow the typical approach: use basic knowledge of the system on which the ground truth is obtained to prune the parameter set and consider the most likely relevant parameters. For instance, we know that the nodes are homogeneous, and thus use the same parameter values for all nodes; and we know that all control messages are small and that our platform's network has low latency and high bandwidth, thus justifying using the default 0-byte value since control messages have negligible impact on the execution.

5.3 Instantiating the Automated Calibration

In this section we apply the methodology in Section 3 to the simulator that implements the highest level of detail, computing calibrations using 48 cores of a dedicated Intel Xeon Gold 2.8GHz CPU.

5.3.1 Parameter Ranges.

We use broad parameter ranges: network and disk bandwidths are 2^x bits per second and core speeds are 2^x ops per second for $20 \leq x \leq 40$, network latencies are between 0ms and 10ms, overheads are between 0s and 20s, and the maximum number of concurrent I/O operations at a disk is between 1 and 100.

5.3.2 Loss Functions and Algorithms.

The ground-truth data reports the *makespan* (i.e., overall execution time) and the execution times of each individual task. For workflow i, let m_i, resp. $\hat{m}_i$, denote the ground-truth, resp. simulated, makespan; and let $t_{i,j}$, resp. $\hat{t}_{i,j}$, denote the ground-truth, resp. simulated, execution time of task j in workflow i. For workflow i, let $e_i = |(m_i - \hat{m}_i)/m_i|$ denote the error on the makespan, and $e_{i,j} = |(t_{i,j} - \hat{t}_{i,j})/t_{i,j}|$ denote the error on a task's execution time. Among the many possible options for defining a loss function we consider:

- $\mathcal{L}_1$: $\text{avg}_i(e_i)$
- $\mathcal{L}_2$: $\max_i(e_i)$
- $\mathcal{L}_3$: $\text{avg}_i(e_i + \text{avg}_j(e_{i,j}))$
- $\mathcal{L}_4$: $\max_i(e_i + \text{avg}_j(e_{i,j}))$
- $\mathcal{L}_5$: $\text{avg}_i(e_i + \max_j(e_{i,j}))$
- $\mathcal{L}_6$: $\max_i(e_i + \max_j(e_{i,j}))$

These loss functions minimize average and/or maximum errors. $\mathcal{L}_1$ and $\mathcal{L}_2$ only consider the makespan error, hence do not account for the temporal structure of the execution. $\mathcal{L}_3$ to $\mathcal{L}_6$ combine the makespan error and the task execution time errors in various ways that a user may employ.

As explained in Section 3, we use synthetic benchmarking to compare the calibration error of different loss functions. Table 3 shows calibration error for different algorithm/loss combinations (lower values are better). Overall, BO-GP outperforms RAND. Using BO-GP with the $\mathcal{L}_1$ loss function leads to the best calibration error (even though $\mathcal{L}_1$ is the simplest loss function). Consequently, in all that follows, we use BO-GP with $\mathcal{L}_1$.

5.3.3 Calibration Time Budget.

As expected, the longer the time budget the better the calibration and/or the better the use of a larger training dataset. We picked a 24-hour time budget for our experiments, which is sufficient for the loss function to converge. For instance, Figure 1 shows loss vs. time for a particular workflow (similar behavior is seen for other workflows). The loss improves rapidly in the first two hours, and only improves marginally afterwards.
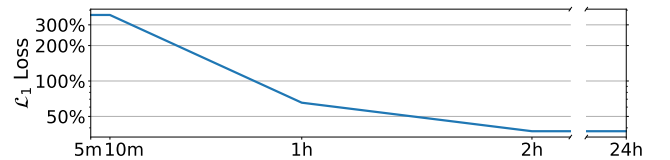
Figure 1: Loss value vs. time when computing a calibration using all ground-truth data for the Epigenomics workflow.
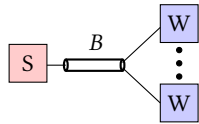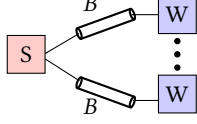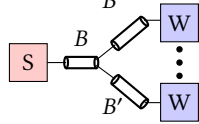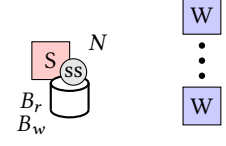
Table 2: Level of detail options for the simulators used in Case Study #1.

Table 3: Calibration error vs. algorithm and loss function.

Alg. \ Loss	$\mathcal{L}_1$	$\mathcal{L}_2$	$\mathcal{L}_3$	$\mathcal{L}_4$	$\mathcal{L}_5$	$\mathcal{L}_6$
RAND	541.24	111.43	610.82	883.40	130.55	883.40
BO-GP	30.96	935.10	935.10	89.76	89.76	89.76

5.4 Picking the Level of Detail

In practice a user would calibrate a simulator, decide whether a different level of detail is more desirable, implement a new simulator, and repeat. For the purpose of this case study we instead calibrate all 12 simulator versions. These calibrations should be computed based on a subset of the ground-truth data (the "training dataset") and their simulation accuracy should be evaluated on the remaining data (the "testing dataset"). For each workflow application, our ground-truth data includes executions for four, resp. five, distinct numbers of workers, resp. tasks. We define the testing dataset as the "large" executions, i.e., all executions for the largest number of workers and/or the largest number of workflow tasks (excluding executions for the smallest number of workers and the smallest number of tasks). For instance, for the 1000Genome workflow, ground-truth executions are for 1, 2, 4, and 6 workers; and for 54, 81, 108, 162, and 270 tasks. Our testing dataset comprises all executions on 6 workers with 81 or more tasks, and all executions with 270 tasks on 2 or more workers. In this section, we define the training dataset as all workflow executions for the second largest number of workers and the second largest number of tasks (e.g., for 1000Genome these

are executions for 4 workers and 162 tasks). We study the impact of the training dataset in Section 5.5.
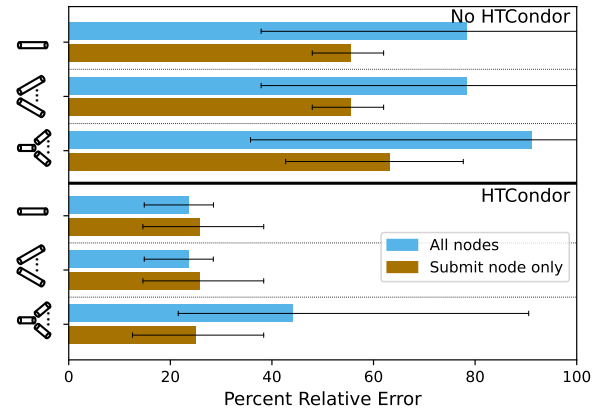
Figure 2: Percent relative error between simulated and ground-truth makespans (bars show averages values over all workflows; error bars show min and max values). The top, resp. bottom, half of the figure is for not simulating, resp. simulating, HTCondor. Each pair or bars is for one of the three options for simulating the network (depicted as simplified network diagrams on the vertical axis). Each bar in a pair is for a different option for simulating the storage system, either on the submit node only or on all nodes.

Figure 2 shows, for all 12 calibrated simulator versions, the relative percentage error between simulated and ground-truth makespans. This is the metric that most users would likely care about (and also happens to also be the loss value in percentage). The first observation is that simulating HTCondor is crucial. This is because the simulated HTCondor component simulates overheads that occur at different phases of a task's execution. Considering only the bottom half of the figure, we find that using a relatively low level of detail for the network topology is sufficient. This is because the real-world network is sufficiently provisioned to support all concurrent data transfers in the ground-truth workflow executions. As a result, the one-link and the star topology leads to equivalent results. Our more complex topology with a shared link and a dedicated link in series does worse. This may seem surprising because setting the bandwidth of the shared link, resp. of the dedicated links, to a high bandwidth effectively yields a star, resp. one-link, topology. But this more complex topology increases the dimensionality, and thus the difficulty, of the simulation calibration problem without bringing any accuracy benefit (at least given our available ground truth). Finally, we find that simulating a storage system on all nodes brings only marginal benefit over simulating a storage system on the submit node only. In the real-world deployment there is a storage system at all nodes. But simulating storage only on the submit node provides a reasonable approximation, at least in the scope of our available ground truth.

In the end, for this case study, our approach makes it possible to conclude that the best accuracy is achieved when simulating (i) HTCondor; (ii) a one-link topology; and (iii) a storage system at all nodes. Because all considered simulators are calibrated to the best of their ability, and can thus be compared soundly, a user can pick the simulator that maximizes their utility. All simulators achieve comparable simulation speed in this case study, and most users would likely pick the most accurate simulator. This simulator achieves error around 20% and relatively low variance across workflows, meaning that it can be used reliably to compare simulated executions and draw conclusions regarding real-world systems. One source of simulation inaccuracy is that the simulator does not reproduce the ground-truth task schedules exactly. To do so, the simulator would have to implement the exact same scheduling algorithms as that in Pegasus and HTCondor, and use the exact same data structures, which we have not done in this case study.

To assess the value added by our proposed methodology, we consider an approach in which one would simply use the simulator implemented at the lowest level of detail and set all parameter values based on available hardware specifications (i.e., documented on the Chameleon Cloud website and/or inspected directly on the nodes). This is likely in line with what authors do when they do not mention calibration (e.g., as in more than 70% of the publications reviewed in [36]). Using this approach, the average percent relative error between simulated and ground-truth makespans ranges from 110% to 1,412% over the five workflow applications, i.e., orders of magnitude worse than our automatically calibrated simulators.

5.5 Use of Ground-Truth Data

A simulator should be calibrated with respect to a sufficiently large and diverse set of ground-truth executions for the computed calibration to yields low loss on the testing dataset. Obtaining ground-truth data has a cost as it requires labor, time, and hardware resources. Also, using more ground-truth data increases the evaluation time of the loss function (due to a larger number of simulator invocations), which in turn can impede the exploration of the calibration search space. There is thus an incentive to compute calibrations based on only a few and/or small-scale executions. The risk is to overfit to these executions, obtaining a calibration that is "correct for the wrong reasons" and thus non-generalizable to other executions.

In the previous section, the training dataset is the executions with the second largest number of workers and number of tasks. In this section we consider other options using two different schemes: (i) "single-sample" training based on executions for a single number of workers (n) and a single number of tasks (m); and (ii) "rectangular-sample" training based on executions for all numbers of workers $\leq n$ and for all numbers of tasks $\leq m$. Given our available ground truth, for a given workflow this yields 27 different options for the training dataset. Obtaining the ground-truth data for each of these options has a resource cost, with higher cost for more workflow executions, more workers, and/or longer workflow executions. We measure the cost of obtaining the ground-truth data for a particular training dataset as the sum, over all included workflow executions, of the number of workers multiplied by the makespan, in seconds.
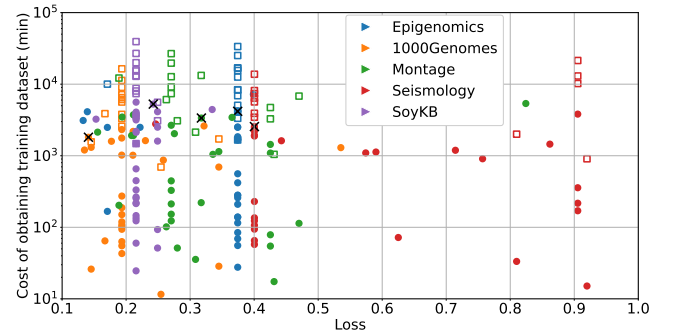
Figure 3: Training dataset cost vs. loss, for each workflow. For each workflow, data points are shown for the single-sample scheme as filled circles and for the rectangular-sample scheme as unfilled squares. Crosses indicate the training datasets used in the previous section.

Figure 3 shows results as a scatter plot where the horizontal axis is the achieved loss and the vertical axis is the cost of obtaining the training data, for all workflows. Although the training datasets used in the previous section were not the best choices, they achieve relatively low loss at relatively low cost. An interesting observation from these results is that using larger training datasets (in the number of data points, as in the rectangular-sample scheme, or in the scale of the executions for these data points) can be detrimental. This may seem counter-intuitive, but is explained as follows. A simulation for a single number of workers and single number of tasks does exert all components of the simulated systems with many simulated task executions, data transfers, and I/O

operations. The executions for other numbers of workers and/or tasks are not necessarily qualitatively different, in which case there is little new behavior to learn. Including these executions leads to an unnecessarily large training dataset. This is detrimental because the loss function evaluation involves simulating the execution of all workflows in the training dataset. For a given time budget, the optimization algorithm will thus perform fewer iterations and may produce a worse calibration. Another observation is that using the cheapest (running the smallest workflow configuration on a single worker), and thus necessarily least diverse, training datasets is a poor choice. Save for the SoyKB workflow, the cheapest training dataset for each workflow in Figure 3 is among the ones with the highest loss. We conclude that, for this case study, a training dataset with executions for a single number of workers > 1 and single number of tasks is sufficient.

A way to further reduce the training dataset cost is to, for each worker count and task count, execute fewer workflow configurations. As seen in Table 1, our ground-truth data spans a range of sequential work and data footprint values. We computed calibrations using only executions for one sequential work value and one data footprint value. In more than 98% of the cases the resulting loss on the testing dataset becomes significantly, and sometimes up to an order of magnitude, larger. Expectedly, the worst results are when the training dataset only includes executions with zero sequential work and/or zero data footprint, meaning that some components of the system are never simulated. We conclude that the training dataset must include executions with diverse data to compute volume ratios. Otherwise, calibrations are overfit to a single such ratio and thus non-generalizable.

Yet another way to reduce the training dataset cost is to employ simple benchmarks, such as the chain or forkjoin workflows described in Section 5.1. We computed calibrations with a training dataset that contains only chain executions, forkjoin executions, or both. The ground-truth data for real-world workflows is used as the testing dataset. Using only chain executions increases the loss by more than one order of magnitude due to the training dataset not including any parallel task executions. Using only forkjoin executions leads to loss increases between 1.2x and 3.5x depending on the workflow. Computing calibrations based on both chain and forkjoin executions leads to worse results, due to the loss function evaluation being more costly. Overall, the use of simple benchmark ground-truth for calibrating simulations of real-world applications is attractive but has a non-negligible negative impact on accuracy, at least in the scope of this case study.

6 Case Study #2: Message Passing Applications

Many researchers have used simulation to investigate the performance of MPI (Message Passing Interface) applications, relying on one of the many simulation frameworks developed for this purpose [7, 19, 22, 27, 31, 38]. In this section we apply our approach to a simulator that was developed to investigate MPI performance on a particular HPC cluster. All relevant datasets and software are available at [3].

6.1 Ground Truth

The ground truth that was made available to us is from runs of the Intel MPI Benchmarks (IMB) [30] on the pre-exascale Summit leadership class supercomputer at the Oak Ridge National Laboratory. The IMB measures the performance of various MPI point-to-point communication functions and patterns for ranges of message sizes. Specifically, the collected ground-truth data consists of execution logs of the PingPing, PingPong, BiRandom, and Stencil IMB benchmarks with 2^x-byte messages, for $x \in \{10, 11, \ldots, 22\}$, on 128, 256, and 512 compute nodes.

Summit is an IBM system with ~4,600 compute nodes, each equipped with two IBM POWER9 processors, for a total of 42 CPU cores, and with six NVIDIA Tesla V100 GPUs. Summit's interconnect uses a non-blocking three-level Fat-Tree topology: Level one switches connect 18 nodes in each cabinet along with 18 director switches comprising 36 level two and 18 level three switches to connect cabinets together. On platforms like Summit, the bulk of the performance comes from the use of the multiple GPUs at each node. The usage of CPUs is often limited to the control of the execution flow of the application, to send and retrieve data to and from GPUs, and to manage inter-node data exchanges using MPI. On Summit, with six GPUs per node, often only six out the 42 available CPU cores are used. The ground-truth executions match this practice by running the IMB with six MPI ranks per node.

6.2 Simulator Versions

Our simulator uses SMPI [19], which makes it possible to compile and simulate unmodified MPI programs. SMPI takes as input a specification of a simulated hardware platform and executes the MPI application code as is, with each MPI rank executing as a thread. Each MPI call is intercepted and its duration is simulated, while each block of code between two calls is timed to simulate computation delays. The simulator takes in three arguments for specifying the level of detail for the simulation of three particular components: (i) the network; (ii) the compute nodes; and (iii) the adaptive communication protocol.

We consider four options for simulating the network topology: (i) a single shared backbone link; (ii) a single shared backbone link and dedicated links to each compute nodes; (iii) a 4-ary tree network; and (iv) a fat tree topology with several layers of switches, which corresponds to the network topology used on Summit.

We consider two options for simulating the compute nodes: (i) a multi-core node with NIC, which abstracts away architectural details; and (ii) a two-socket node where the sockets communicate via an X-Bus SMP bus, and are each connected to the NIC via a PCIe bus, which is closer to the real architecture.

MPI implementations use different communication protocols for different message sizes for performance reasons (e.g., switching from eager to rendez-vous mode), which can be modeled as a multiplicative factor applied to data transfer rates [19]. Results in [19] indicate two such message sizes, or change points. We consider two options for simulating this adaptive protocol: (i) protocol changes occur at two known change points (experiments can be conducted to determine the change points empirically), leading to three bandwidth factors to calibrate; and (ii) the same model but where the two change points are unknown and must thus be calibrated (which

removes the need to conduct extra experiments but increases the dimensionality of the calibration problem).

Table **??** depicts the above options and shows the parameters to calibrate, which lead to $4 \times 2 \times 2 = 16$ simulator versions.

6.3 Instantiating the Automated Calibration

In this section we apply the methodology in Section 3 to the simulator that implements the highest level of detail, computing calibrations on 48 cores of a dedicated Intel Xeon Gold 2.8GHz CPU.

6.3.1 Parameter Ranges.
We use broad parameter ranges: for all bandwidth, latencies, and compute speeds, we use min, resp. max, values that are at least one order of magnitude lower, resp. higher, than the actual hardware specification of Summit.

6.3.2 Loss Functions and Algorithms.
For each benchmark and for a given message size, our ground-truth data consists of data transfer rate measurements, or samples, for multiple repeated executions. Due to platform noise, there is variance in these samples. Our simulator, instead, produces a single data transfer rate value since SMPI simulations are deterministic and repeatable by design. We need to quantify how representative this simulated value is of the set of measured samples. To do so we use the *explained variance*, which is defined as a/b, where a is the L1 distance between the samples and the model value, and b is the L1 distance between the measured samples and their mean. The lower (i.e., closer to 1) the explained variance, the more closely the simulated value matches the measured sample. Let $ev_{i,j}$ denote the explained variance between the ground-truth data transfer rates and the simulated data transfer rate for benchmark i executed with message size j. We consider four loss functions defined based on the explained variance:

- $\mathcal{L}_1$: $\text{avg}_i(\text{avg}_j(ev_{i,j}))$
- $\mathcal{L}_2$: $\text{avg}_i(\text{max}_j(ev_{i,j}))$
- $\mathcal{L}_3$: $\text{max}_i(\text{avg}_j(e_{i,j}))$
- $\mathcal{L}_4$: $\text{max}_i(\text{max}_j(e_{i,j}))$

Table 5 shows the calibration error, computed using the synthetic benchmarking technique in Section 3. Results are shown for each combination of algorithm and loss function, when computing (and evaluating) calibrations using all generated synthetic ground-truth data for the PingPing, PingPong, and BiRandom benchmarks. This error measure may, however, be misleading due to the bandwidth factors used to simulate the adaptive message-passing protocol: calibrations that differ only in bandwidths and multiplicative bandwidth factors can produce the same simulated execution. That is, simulating a link with bandwidth B with multiplicative factor α is the same as simulating a link with bandwidth αB with multiplicative factor 1. For this reason, Table 5 also shows the relative absolute error between real-world and simulated transfer rates, averaged over all benchmarks and data sizes. Overall, the best combination, which we use in all that follows, is the BO-GP algorithm with the $\mathcal{L}_1$ loss function. This is similar to our finding in the previous case study (i.e., use Bayesian Optimization with a simple loss function).

6.3.3 Calibration Time Budget.
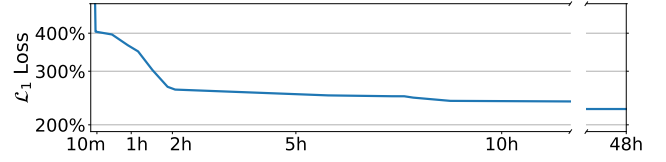We use a 48-hour time budget, which is sufficient for the loss value to converge, as seen in the example in Figure 4.

Figure 4: Loss value vs. time when computing a calibration using all ground-truth data for 128 compute nodes.
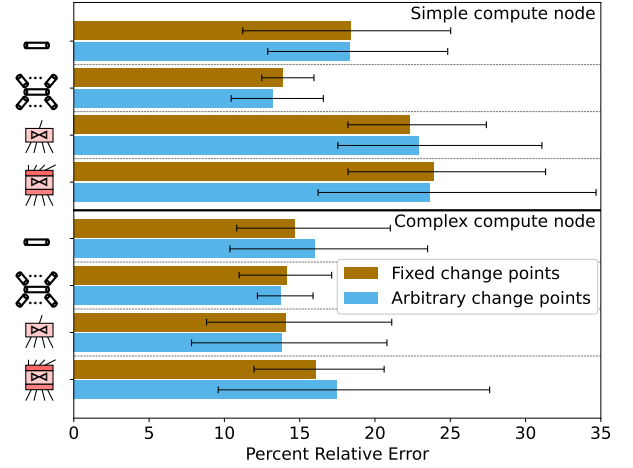

Figure 5: Percent relative error between simulated and ground-truth data transfer rates (bars show average values over all benchmarks; error bars show min and max values). The top, resp. bottom, half of the figure is for simulating a simple, resp. complex, compute node. Each pair of bars is for one of the four options for simulating the network topology (depicted as simplified network diagrams on the vertical axis). Each bar in a pair is for a different option for simulating the adaptive MPI protocol, with either fixed or arbitrary change points.

6.4 Picking the Level of Detail

In this section we compare the accuracy of all 16 simulators, each calibrated using the method instantiated in the previous section. As noted in the previous case study, in practice a user would instead incrementally implement, calibrate, and evaluate simulators. We present overfitting results, i.e., both the training and the testing datasets consist of 128-node ground-truth executions for PingPing, PingPong, and BiRandom. We study the impact of the training dataset in Section 6.5.

Figure 5 shows relative percentage error between simulated and ground-truth data transfer rates. This is an easier accuracy metric to interpret than the loss value, and what most users would measure. The first observation is that all simulators exhibit relatively similar results, more so than in the previous case study, with average relative errors between 13% and 24%. Simulating complex, rather than simple, compute nodes is better in most cases. The one exception is when simulating simple compute nodes and a backbone topology with individual links (i.e., the second pair of bars in the top half of
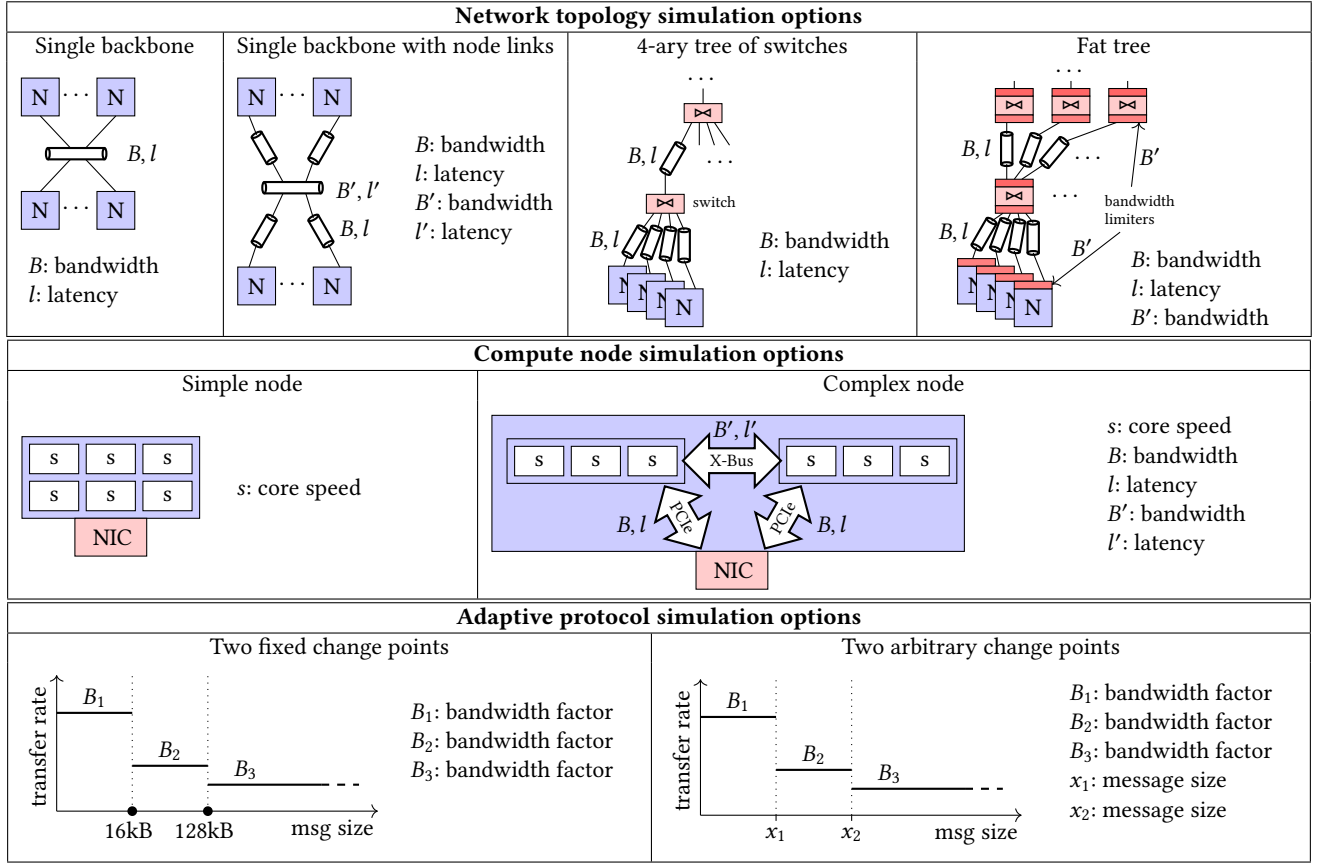
Table 4: Level of detail options for the simulators users in the case study in Section 6.

Table 5: Calibration error and average relative transfer rate error vs. loss function.

Metric	$\mathcal{L}_1$	$\mathcal{L}_2$	$\mathcal{L}_3$	$\mathcal{L}_4$
RAND				
Calibration error	60.96	58.17	43.80	70.74
Rel. avg. transfer rate error	0.05	0.10	0.09	0.22
BO-GP				
Calibration error	1.87	126.65	31.58	47.11
Rel. avg. transfer rate error	0.03	0.12	0.11	0.17

the figure). This particular case happens to lead to average error as low as the best simulator that simulates complex compute nodes (but a higher variance). In terms of MPI protocol simulation, the differences are small in terms of averages. But using fixed, rather than arbitrary, change points, leads to lower variance in most cases. The increase in the level of detail due to non-fixed change points is not worthwhile, at least given our calibration time budget. Finally, we see that the best results are achieved when simulating a backbone topology with individual links, which seems to strike a good compromise between calibration dimensionality and potential accuracy. Simulating the network at a higher level of detail, using

a 4-ary tree or a fat-tree topology, leads to worse results in terms of average and/or variance.

The best accuracy is achieved when simulating (i) a backbone with individual links and (ii) an adaptive MPI protocol with fixed change points. In these conditions, simulating a complex compute node brings almost no benefit compare to simulating a simple compute node. Like in the previous case study, all simulators achieve similar simulation speed. Given the results in Figure 5, a user would most likely opt for simulating a simple compute node, since it is sufficient to achieve good results an requires less simulator code.

The most accurate simulators achieve errors below 15%, with relatively low variance across all benchmarks. As in the previous case study, we have performed a straightforward calibration that a user may do based on Summit's specifications, using the simulator implemented at the lowest level of detail. The average percent relative error between simulated and ground-truth data transfer rates ranges from 91% to 97% over the three benchmarks.

6.5 Use of Ground Truth Data

As in the previous case study, we explore the impact of using different training datasets for computing calibrations, to determine the extent to which a calibrated simulator produces results that generalizes beyond the ground truth. We consider generalization

to (i) different benchmark types and (ii) different execution scales, using the simulator that implements the highest level of detail. First, we simulate the 128-node execution of the Stencil application benchmark using a calibration computed based on the 128-node executions of the BiRandom, PingPing, and PingPong benchmarks. We find that simulated execution achieves a relative error (averaged over all message sizes) of 58.8%. By contrast, using a calibration based on the Stencil ground-truth data, the error becomes 28.6%. Second, we simulate the 256- and 512-node execution of each benchmark using a calibration computed based on their 128-node executions. We observe significant increases in simulation error for all benchmarks. For instance, for the BiRandom benchmark, while error averaged over all message sizes is 15.2% when simulating 128-node executions, when simulating 256- and 512-node executions the error becomes 30.8% and 59.4%, respectively.

These results show that the calibrated simulator does not generalize well beyond the ground-truth data. It may still be useful for some purposes (e.g., studying the impact of the network link bandwidth on particular benchmark executions at particular scales). But overall, this is a negative result for this simulator: it cannot fulfill its intended purpose of studying MPI performance scaling for the target HPC cluster, even for simple point-to-point communications. Perhaps the simulator does not implement a sufficiently high level of detail. But we actually suspect that information on how the ground-truth data was obtained is incomplete and/or inaccurate, leading to the simulated executions to qualitatively differ from the ground-truth executions. Regardless, as far as this work is concerned, this is positive result for our methodology: automatically calibrating the simulator to the best of its ability with respect to the available ground truth makes it possible to systematically evaluate its intrinsic accuracy, and to reach the above (negative) result.

7 Conclusion

Evaluating the intrinsic accuracy of simulators of PDC systems requires that these simulators be well-calibrated. Because simulation calibration is labor-intensive, it must be automated. We have proposed a methodology for instantiating the simulation calibration process and a framework for automating this process. This approach is general and we have demonstrated its usefulness via two case studies representative of current simulation-driven research. In particular, these case studies show that our approach makes is possible to make rational decisions for picking appropriate levels of detail when implementing simulators.

Conclusions regarding the most appropriate level of detail may be use case-specific. However, many researchers use simulation to study similar scenarios, often using the same simulation frameworks or simulators (e.g., batch-scheduling using Alea [33] or Batsim [21] and data from the Parallel Workload Archive [40], scientific workflows using WRENCH [11] with execution logs from WfCommons [16], or cloud computing using CloudSim [8] with ground-truth data from Google [26]). We thus expect some conclusions to generalize across related use cases within the same PDC domain. In future work, we will perform other case studies to verify this expectation. The ultimate goal for our approach, and its implementation, is not only to improve simulation-driven research,

but also to give rise to guidelines regarding which simulation level of detail should be used in particular PDC domains.

References

[1] Jung Ho Ahn, Sheng Li, Seongil O, and Norman Jouppi. 2013. McSimA+: A Manycore Simulator with Application-Level+ Simulation and Detailed Microarchitecture Modeling. In *Proc. of the IEEE Int. Symp. on Performance Analysis of Systems and Software.* 74–85.

[2] A. Al-Haboobi and G. Kecskemeti. 2023. Developing a Workflow Management System Simulation for Capturing Internal IaaS Behavioural Knowledge. *Journal of Grid Computing* 21, 2 (2023).

[3] Anonymous. 2025. Reproducibility Material. https://figshare.com/s/35a8ae457d5b713d2fe5. Will be provided as a full-fledge AD/AE appendix, as required if paper is accepted.

[4] Malcolm Atkinson, Sandra Gesing, Johan Montagnat, and Ian Taylor. 2017. Scientific Workflows: Past, Present and Future. *Future Generation Computer Systems* 75 (2017), 216–227.

[5] Peraketh Benjamin, Madhav Erraguntla, Dursun Delen, and Richard Mayer. 1998. Simulation Modeling at multiple Levels of Abstraction. In *Proc. of the Winter Simulation Conf.*, Vol. 1. 391–398.

[6] Rajkumar Buyya and Manzur Murshed. 2002. 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 (2002), 1175–1220.

[7] Swen Böhm and Christian Engelmann. 2011. xSim: The extreme-scale simulator. In *Proc. of the Int. Conf. on High Performance Computing and Simulation.* 280–286.

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

[9] Trevor Carlson, Wim Heirman, and Lieven Eeckhout. 2011. Sniper: Exploring the Level of Abstraction for Scalable and Accurate Parallel Multi-Core Simulation. In *Proc. of the Int. Conf. for High Performance Computing, Networking, Storage and Analysis.* 1–12.

[10] Chris Carothers, David Bauer, and Shawn Pearce. 2002. ROSS: A High-Performance, Low Memory, Modular Time Warp System. *J. Parallel and Distrib. Comput.* 62, 11 (2002), 1648–1669.

[11] Henri Casanova, Rafael Ferreira da Silva, Ryan Tanaka, Suraj Pandey, Gautam Jethwani, Spencer Albrecht, James Oeth, and Frédéric Suter. 2020. Developing Accurate and Scalable Simulators of Production Workflow Management Systems with WRENCH. *Future Generation Computer Systems* 112 (2020), 162–175.

[12] Henri Casanova, Arnaud Giersch, Arnaud Legrand, Martin Quinson, and Frédéric Suter. 2025. Lowering entry barriers to developing custom simulators of distributed applications and platforms with SimGrid. *Parallel Comput.* 123 (2025).

[13] Chameleon 2025. Chameleon Cloud. https://www.chameleoncloud.org.

[14] W. Chen and E. Deelman. 2012. WorkflowSim: A Toolkit for Simulating Scientific Workflows in Distributed Environments. In *Proc. of the 8th IEEE Intl. Conf. on E-Science.* 1–8.

[15] Leonardo Chwif, Marcos Ribeiro Pereira Barretto, and Ray Paul. 2000. On Simulation Model Complexity. In *Proc. of the Winter Simulation Conf.*, Vol. 1. 449–455.

[16] Taina Coleman, Henri Casanova, Loïc Pottier, Manav Kaushik, Ewa Deelman, and Rafael Ferreira da Silva. 2022. WfCommons: A Framework for Enabling Scientific Workflow Research and Development. *Future Generation Comp. Sys.* 128 (2022), 16–27.

[17] Ewa Deelman, Rafael Ferreira da Silva, Karan Vahi, Mats Rynge, Rajiv Mayani, Ryan Tanaka, Wendy Whitcup, and Miron Livny. 2021. The Pegasus workflow management system: Translational computer science in practice. *Journal of Computational Science* 52 (2021).

[18] 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. 2015. Pegasus: a Workflow Management System for Science Automation. *Future Generation Computer Systems* 46 (2015), 17–35.

[19] Augustin Degomme, Arnaud Legrand, George Markomanolis, Martin Quinson, Mark Stillwell, and Frédéric Suter. 2017. Simulating MPI applications: the SMPI approach. *IEEE Trans. on Parallel and Distributed Systems* 18, 8 (2017), 2387–2400.

[20] Ciprian Dobre, Florin Pop, and Valentin Cristea. 2011. New Trends in Large Scale Distributed Systems Simulation. *Journal of Algorithms & Computational Technology* 5, 2 (2011), 221–257.

[21] Pierre-François Dutot, Michael Mercier, Millian Poquet, and Olivier Richard. 2016. Batsim: a Realistic Language-Independent Resources and Jobs Management Systems Simulator. In *Proc. of the 20th Workshop on Job Scheduling Strategies for Parallel Processing.*

[22] C. Engelmann. 2014. Scaling To A Million Cores And Beyond: Using Light-Weight Simulation to Understand The Challenges Ahead On The Road To Exascale. *Future Generation Computer Systems* 30 (2014), 59–65.

[23] Gilberto Flores, Marcos Paredes-Farrera, Emmanuel Jammeh, Martin Fleury, and Martin Reed. 2003. OPNET Modeler and Ns-2: Comparing the Accuracy of Network Simulators for Packet-Level Analysis Using a Network Testbed. *WSEAS Trans. on Computers* 2, 3 (2003).

[24] Richard Fujimoto. 1990. Parallel Discrete Event Simulation. *Commun. ACM* 33, 10 (1990), 30–53.

[25] Pablo Garrido, Manuel Malumbres, and Carlos Calafate. 2008. Ns-2 vs. OPNET: A Comparative Study of the IEEE 802.11e Technology on MANET Environments. In *Proc. of the 1st Int. Conf. on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops.* 1–10.

[26] Google Workload Traces 2025. Google Workload Traces. https://github.com/google/cluster-data.

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

[28] HTCondor 2023. The HTCondor Software Suite. https://htcondor.org.

[29] Philipp Hurni and Torsten Braun. 2009. Calibrating Wireless Sensor Network Simulation Models with Real-World Experiments. In *Proc. of the 8th Int. IFIP-TC 6 Networking Conf. (Lecture Notes in Computer Science, Vol. 5550).* Springer, 1–13.

[30] IMB 2021. Intel MPI Benchmarks User Guide. https://www.intel.com/content/www/us/en/docs/mpi-library/user-guide-benchmarks/2021-8/overview.html.

[31] C. L. Janssen, H. Adalsteinsson, S. Cranford, J. P. Kenny, A. Pinar, D. A. Evensky, and J. Mayo. 2010. A simulator for large-scale parallel architectures. *International Journal of Parallel and Distributed Systems* 1, 2 (2010), 57–73.

[32] Gabor Kecskemeti. 2015. DISSECT-CF: A Simulator to Foster Energy-Aware Scheduling in Infrastructure Clouds. *Simulation Modelling Practice and Theory* 58 (2015), 188–218.

[33] Dalibor Klusacek, Mehmet Soysal, and Frédéric Suter. 2019. Alea - Complex Job Scheduling Simulator. In *Proc. of the 13th Int. Conf. on Parallel Processing and Applied Mathematics (Lecture Notes in Computer Science, Vol. 12044).* 217 – 229.

[34] Johannes Lessmann, Peter Janacik, Lazar Lachev, and Dalimir Orfanus. 2008. Comparative Study of Wireless Network Simulators. In *Proc of the 7th Int. Conf. on Networking.* 517–523.

[35] Fabian Mastenbroek, Georgios Andreadis, Soufiane Jounaid, Wenchen Lai, Jacob Burley, Jaro Bosch, Erwin van Eyk, Laurens Versluis, Vincent van Beek, and Alexandru Iosup. 2021. OpenDC 2.0: Convenient Modeling and Simulation of Emerging Technologies in Cloud Datacenters. In *Proc. of the 21st IEEE/ACM Int. Symp. on Cluster, Cloud and Internet Computing.* 455–464.

[36] Jesse McDonald, Maximilian Horzela, Frédéric Suter, and Henri Casanova. 2024. Automated Calibration of Parallel and Distributed Computing Simulators: A Case Study. In *Proc. of the 25th IEEE Int. Workshop on Parallel and Distributed Scientific and Engineering Computing.* 1026–1035.

[37] Julien Monniot, François Tessier, Henri Casanova, and Gabriel Antoniu. 2024. Simulation of Large-Scale HPC Storage Systems: Challenges and Methodologies. In *Proc. of the 31st IEEE Int. Conf. on High Performance Computing, Data, and Analytics.* 1–11.

[38] Misbah Mubarak, Christopher Carothers, Robert Ross, and Philip Carns. 2017. Enabling Parallel Simulation of Large-Scale HPC Network Systems. *IEEE Trans. on Parallel and Distributed Systems* 28, 1 (2017), 87–100.

[39] Simon Ostermann, Kassian Plankensteiner, Radu Prodan, and Thomas Fahringer. 2010. GroudSim: An Event-Based Simulation Framework for Computational Grids and Clouds. In *Proc. of the Euro-Par Parallel Processing Workshops.* 305–313.

[40] Parallel Workloads Archive 2025. Parallel Workloads Archive. https://www.cs.huji.ac.il/labs/parallel/workload.

[41] Alejandro Rico, Felipe Cabarcas, Carlos Villavieja, Milan Pavlovic, Augusto Vega, Yoav Etsion, Alex Ramirez, and Mateo Valero. 2012. On the Simulation of Large-Scale Architectures Using Multiple Application Abstraction Levels. *ACM Trans. on Architecture and Code Optimization* 8, 4 (2012), 1–20.

[42] Sashko Ristov, Mika Hautz, Christian Hollaus, and Radu Prodan. 2022. SimLess: Simulate Serverless Workflows and Their Twins and Siblings in Federated FaaS. In *Proc. of the 13th Symp. on Cloud Computing.* 323–339.

[43] Stewart Robinson. 2011. Choosing the Right Model: Conceptual Modeling for Simulation. In *Proc. of the Winter Simulation Conf.* 1423–1435.

[44] Stewart Robinson and Roger Brooks. 2024. Assumptions and simplifications in discrete-event simulation modelling. *Journal of Simulation* (2024), 1–18.

[45] scikit-optimize 2025. scikit-optimize: Sequential model-based optimization in Python. https://scikit-optimize.github.io/stable/.

[46] SST-Macro 2024. SST/macro 14.1: User's Manual. https://raw.githubusercontent.com/sstsimulator/sst-macro/refs/heads/master/manual-sstmacro-14.1.pdf.

[47] Chih-Li Sung and Rui Tuo. 2024. A Review on Computer Model Calibration. *WIREs Computational Statistics* 16, 1 (2024), e1645.

[48] Michael Tighe, Gaston Keller, Michael Bauer, and Hanan Lutfiyya. 2012. DCSim: A Data Centre Simulation Tool for Evaluating Dynamic Virtualized Resource Management. In *Proc. of the Workshop on Systems Virtualization Management.* 385–392.

[49] Irfan Uddin. 2015. Multiple Levels of Abstraction in the Simulation of Microthreaded Many-Core Architectures. *Open Journal of Modelling and Simulation* 3 (2015), 159–190.

[50] WfCommons 2025. The WfCommons Project. https://wfcommons.org.

[51] wrench [n. d.]. WRENCH: Workflow Management System Simulation Workbench. http://wrench-project.org.

[52] Urooj Yousuf Khan, Tariq Rahim Soomro, and Muhammad Nawaz Brohi. 2022. iFogSim: A Tool for Simulating Cloud and Fog Applications. In *Proc. of the Int. Conf. on Cyber Resilience.* 01–05.