Automated Calibration of Parallel and Distributed Computing Simulators: A Case Study

Jesse McDonald
University of Hawai'i at Mānoa
Honolulu, HI, USA
jamcd@hawaii.edu

Maximilian Horzela
Karlsruhe Institute of Technology
Karlsruhe, Germany
maximilian.horzela@kit.edu

Frédéric Suter
Oak Ridge National Laboratory
Oak Ridge, TN, USA
suterf@ornl.gov

Henri Casanova
University of Hawai'i at Mānoa
Honolulu, HI, USA
henric@hawaii.edu

***Abstract*—Many parallel and distributed computing research results are obtained in simulation, using simulators that mimic real-world executions on some target system. Each such simulator is configured by picking values for parameters that define the behavior of the underlying simulation models it implements. The main concern for a simulator is accuracy: simulated behaviors should be as close as possible to those observed in the real-world target system. This requires that values for each of the simulator's parameters be carefully picked, or "calibrated," based on ground-truth real-world executions. Examining the current state of the art shows that simulator calibration, at least in the field of parallel and distributed computing, is often undocumented (and thus perhaps often not performed) and, when documented, is described as a labor-intensive, manual process. In this work we evaluate the benefit of automating simulation calibration using simple algorithms. Specifically, we use a real-world case study from the field of High Energy Physics and compare automated calibration to calibration performed by a domain scientist. Our main finding is that automated calibration is on par with or significantly outperforms the calibration performed by the domain scientist. Furthermore, automated calibration makes it straightforward to operate desirable trade-offs between simulation accuracy and simulation speed.**

***Index Terms*—Simulation of distributed computing platforms and applications, Simulation accuracy and scalability, Simulation calibration.**

I. Introduction

Much Parallel and Distributed Computing (PDC) research relies on experimental results obtained by executing application workloads on PDC platforms. Many published works include results obtained in simulation, either in addition to or as a replacement for results obtained from real-world experiments. Simulation is attractive for several reasons. It makes it possible to explore hypothetical workload and platform configurations. It also yields results that are 100% reproducible and observable. Finally, in most cases, simulation experiments entail significantly less time, labor, carbon footprint, and/or funds than their real-world counterparts. The main concern with simulation is *accuracy*, that is, how well real-world and corresponding simulated executions match.

Consider a PDC system that consists of some software stack used to execute an application workload on a hardware platform, and a simulator of this system. The simulator's underlying *simulation models* can operate at various levels of abstraction. As an example consider a distributed file system whose performance needs to be studied in simulation. A simulator could be developed that simulates the system as a black box server with some stochastic service time (high level of abstraction), or as a set of daemons that perform I/O operations on local disks and coordinate over the network using some distributed algorithm (low level of abstraction). Whenever data is sent to the distributed file system the time necessary for that data transfer could be based on packet-level simulation of network communications (low level of abstraction) or computed as a data size divided by a bandwidth (high level of abstraction). While low levels of abstractions can potentially lead to high simulation accuracy they also typically increase the space- and time-complexity of the simulator, hence the well-recognized trade-off between simulation accuracy and simulation scalability (speed and memory footprint).

Regardless of their levels of abstraction, the simulation models in a PDC simulator all come with configuration *parameters*. These parameters pertain to the hardware platform (e.g., network and disk bandwidths, CPU clock rates, cache sizes), the software stacks (e.g., TCP window size, software overhead to start a virtual machine instance, size of some control messages), and the application workload (e.g., data and compute volumes, task granularity, control/data dependencies). Simulation models can be developed from scratch by simulator developers themselves, who then define the set of relevant parameters. Alternately, simulator developers can use simulation models provided by simulation frameworks designed specifically to ease PDC simulator development [1]–[9].

Different values for the parameters of the simulation models used by a simulator of a target PDC system lead to different simulated executions. It is thus critical that values be chosen that make the simulation is as accurate as possible. Some parameter values may be straightforward to determine, such

as the parameters that define the application workload. But for others, typically those that pertain to the hardware platform, picking good values can be challenging. A seemingly natural approach is to pick values based on knowledge of the PDC system. Unfortunately, these values may not be known to the user (e.g., parameters that define the precise network topology). Furthermore, if the simulator's level of abstraction is high, a single parameter value may not map directly to a single known characteristic of the target system. For instance, if a simulator abstracts a complicated network topology as a single network link with a latency and a bandwidth, it is difficult to come up with reasonable values for these two parameters based on knowledge (assuming this knowledge even exists) of the real-world network topology. A more sound approach is instead to pick parameter values that maximize simulation accuracy with respect to one or more execution scenarios on a real-world system. We term this approach *simulation calibration*. We say that simulation models, or simulators that use these models, have been calibrated if all parameter values have been determined so that simulated executions are as similar as possible to *ground-truth* real-world executions. The hope is that the calibrated simulator will then achieve high accuracy when simulating executions that go beyond the ground-truth executions used to calibrate it (different application workloads, different platform scales, different platform hardware characteristics, etc.).

Simulation calibration ensures that meaningful conclusions can be drawn from simulation results obtained with PDC simulators. Yet, many research works present simulation results without any mention of calibration, perhaps indicating that it was not performed. This may be because these works use simulators developed using some PDC simulation framework, which come with built-in simulation models. These models typically come with default parameter values, and these default values may have been obtained based on calibration for some particular real-world system (or averaged over calibrations for multiple real-world systems). At any rate, there is no guarantee that these default values will be appropriate for all possible scenarios. Some works do mention calibration and give some details about the calibration procedure, which is typically a labor-intensive, at least partially manual, procedure.

We claim that, overall, calibrating simulators of PDC systems is challenging and often not performed (sufficiently or sufficiently thoroughly) in practice. In this work, we verify this claim based on inspection of the literature, and evaluate the potential benefit of automating simulation calibration. More specifically, our contributions include:

- An investigation of the state of the art of the calibration of simulators of PDC systems;
- An instantiation of a general automated simulation calibration procedure with simple algorithms;
- An evaluation of the benefit of automated simulation calibration when compared to a calibration performed by a domain scientist for a production use case.

This paper is organized as follows. Section II reviews related work and attempts to characterize the state of the art of PDC simulator calibration. Section III defines the simulation calibration problem and describes our algorithms. Results are presented in Section IV for a High Energy Physics production use case. Finally, Section V concludes with a brief summary and perspectives on future work.

II. Related Work

A. Simulation of PDC Systems

Many frameworks have been developed for PDC system simulation. Several have garnered sizable user communities and are still actively being maintained at the time of writing [1]–[9]. Different frameworks achieve different compromises between accuracy and scalability by implementing different kinds of simulation models. At one extreme are models that are designed at low levels of abstraction to capture "microscopic" behaviors of hardware/software components, which favors accuracy over scalability (e.g., packet-level network simulation, cycle-accurate CPU simulation, block-level disk simulation). At the other extreme are analytical models that abstract away microscopic behavior and instead attempt to capture "macroscopic" behaviors via empirical mathematical models. While the latter models have lower space- and time-complexity, they must be developed with care so that high accuracy can be achieved [10].

This work is agnostic to the simulation framework used to develop the simulator. However, if the simulation models it provides have inherent inaccuracies (e.g., too high a level of abstraction, implementation/design flaws), calibrating parameters optimally could still lead to low simulation accuracy. In Section IV, we employ a simulator developed using WRENCH [8], [11], which implements high-level simulation abstractions for easy development of simulators of PDC systems. WRENCH itself builds on top of SimGrid [9], [12], which comes with implementations of simulation models that are high-level enough to achieve high simulation scalability. These simulation models have also been thoroughly validated [8], [10], [13]–[21], meaning that they can lead to high simulation accuracy provided their configuration parameter values are chosen appropriately.

B. Calibration of PDC Simulators

The lower a simulation model's level of abstraction, the more directly its parameters map to the characteristics of the system to be simulated. Low-level simulation abstractions are the norm in several fields such as computer architecture and networking. For instance, many published networking research results are obtained using packet-level simulators in which the lifecycle of each individual network packet is simulated via several, and possibly many, discrete events. The parameters of the such simulation models map directly to the physical characteristics of the network links and routers and to the network protocol implementations in the target real-world system to be simulated. It should thus be possible to pick appropriate values for these parameters based solely on the specification of the target system. However, many authors have

TABLE I
EXAMINATION OF 114 RESEARCH PUBLICATIONS IN THE 2017-2022 TIME PERIOD THAT INCLUDE RESULTS OBTAINED WITH SIMGRID.

# Publications that only include simulation results	85		
# Publications that include both simulation and real-world results	29	No comparison thereof	4
		Calibration perhaps performed or at best mentioned	15
		Calibration performed and documented	10

found that doing so is challenging, and that calibration is necessary for achieving high accuracy [22]–[26].

Hundreds of PDC research works have been published that include results obtained with simulators built using various simulation frameworks. Most of these frameworks do not implement low-level simulation models. Instead, they implement simulation models at high levels of abstraction to achieve high scalability, i.e., to make the simulation of large-scale and/or long-running execution scenarios feasible. Because of these high levels of abstraction picking appropriate values for simulation model parameters is even more challenging. That is, there may be no one-to-one correspondence between simulator parameters and system characteristics, meaning that even perfect knowledge of the target system may not be sufficient to pick appropriate parameter values. Instead, parameter values should be determined base on calibration with respect to ground-truth real-world executions.

Assessing the state of the art of calibration of PDC simulators is difficult. But a popular PDC simulation framework, SimGrid [12], maintains a list of research publications that include results obtained using this framework (https://simgrid.org/usages.html). At the time of writing, this list includes 610 publications, 114 of which are peer-reviewed journal or conference/workshop publications for the 2017-2022 6-year time period. As an attempt to assess the state of the art, we have examined these 114 publications in detail to determine how they perform simulator calibration (many authors refer to calibration as "parameter picking" or "parameter tuning"). Our results are summarized in Table I. Of the 114 publications, 85 include only simulation results, which likely indicates that calibration was not performed. In some of these works the goal is only to simulate a simplified model of an abstract system. In others, real-world data is used as input to the simulation, but no comparison with the real-world execution that has generated that data is done (or can be done). In yet other works, the goal is to simulate a system with hardware/software technology that does not (yet) exist. One reason why many papers do not include real-world results is because simulation is often used precisely because such results cannot be obtained in practice. In several of these works, however, the simulation is intended to be representative of real-world systems. If calibration was not performed it is not clear what these systems are.

29 of the 114 works we have reviewed include both simulation and real-world results. We use a broad definition of the term "real-world", which includes not only results obtained on real-world hardware platforms, but also results obtained using emulation and results obtained using low-level simulation (e.g., results obtained using packet-level simulation of networks). 25 of these 29 works perform or allow comparison of simulation and real-world results. 15 of these 25 works either do not detail any calibration procedure or merely mention that picking better simulation parameters improves accuracy. Some of these works, however, present simulation results that exhibit high accuracy, which may indicate that calibration was performed even if not mentioned.

Overall, out of the 114 publications we reviewed only 10 are explicit about performing some calibration and give some details. Half of these describe manual painstaking procedures by which simulation parameter values are picked based on quantitative and qualitative comparisons of real-world and simulation execution logs and metrics, and sometimes on inspecting the source code of the target system's software stacks. The other half do perform similar procedures but also rely on simple statistical techniques (i.e., regressions). It is important to note that, for 8 of these 10 works, the main research contribution is a novel simulation model. Calibration is thus necessary to validate this simulation model. In the end, among the 106 publications that target a non-simulation-related research topic, we found only 2 that performs a solid and documented calibration procedure so as to ensure that simulation results are accurate.

The above discussion indicates that simulator calibration is likely not performed routinely in the PDC field. Parameters may be picked based on best guesses or simply by using the default values for models provided by simulation frameworks. These defaults can come from calibration with respect to some real-world systems available to the developers of the simulation framework (as it is the case with SimGrid). Consequently, published simulation results obtained using default parameter values may be valid for some system configurations, but not necessarily for that of the particular system of interest. Furthermore, not all simulation models are provided by the simulation framework, and custom models are also developed for each particular simulator. These custom models may not come with any (calibrated) default parameter values. Finally, we find that those works that perform simulation calibration typically employ labor-intensive, partially manual, procedures.

All the above provides a strong motivation to automate PDC simulator calibration, which, to the best of our knowledge, has not been reported in the PDC literature. The general ideal of simulation model parameter calibration is of course not new, and has been studied from theoretical standpoints [27]. It is thus not surprising that automated calibration approaches have been proposed in many disciplines [28]–[30].

III. Automated Calibration

A. Problem Statement

We define a *PDC system* as: (i) a hardware platform with compute and I/O resources distributed over a network; (ii) an application workload that consists of compute tasks that use and produce data items; and (iii) a runtime system that is used to execute the workload on the platform. Consider a real-world such system on which the application workload has been executed repeatedly and perhaps for different configurations of the system (e.g., different subsets of the resources, different application workload instances, different runtime system configurations). Each such execution produces a *ground-truth execution trace*, i.e., a log of time-stamped execution events, such as compute task start times and completion times.

Consider now a simulator of the system that implements several simulation models each of which can be configured via parameters. In practice parameters can take many possible values (e.g., the bandwidth of a network link in MBps, the compute speed of a core in GHz, the maximum number of supported concurrent connections outgoing from a data server) or only a few (e.g., a binary value that specifies whether some feature of the runtime system is enabled). The simulator is implemented so that it takes as input a set of parameter values and produces an execution trace that is comparable to a ground-truth execution trace. This comparison is done via a user-defined metric that quantifies the discrepancy between a simulated and a ground-truth execution trace as a measure of simulation accuracy. The simplest simulation accuracy metric is the relative makespan difference, that is, the relative difference between the time elapsed between the start of the first task and the completion of the last task in the ground-truth and in simulation.

Given a PDC system, a set of ground-truth execution traces, a simulator of the system, and a simulation accuracy metric, calibration is the optimization problem that consists in determining the simulator's parameter values that optimize the simulation accuracy metric.

The simulator developer must specify the range of possible values of each parameter. The narrower these ranges the more constrained the search space, but the higher the risk that the best parameter value lies outside that range. Simulator developers specify ranges based on their best guesses and knowledge of the target system, but in practice some parameter ranges could be large. For all results in this work we use a logarithmic representation of each parameter. That is, each parameter with a user-specified range $[a, b]$ is written as 2^x and x is sampled in the interval $[\log_2 a, \log_2 b]$. The rationale is that by sampling values logarithmically, we ensure a bigger diversity of orders of magnitudes within the parameter range. For instance, consider a parameter that describes the bandwidth of some network link. Sampling values of, say, 10 Mbps and 11 Mbps is likely useful, but sampling values of, say, 100,000 Mbps and 100,001 Mbps probably is not.

Evaluating the objective function entails executing the simulator with sets of candidate parameter values, but the simulator's execution time is non-zero and could be relatively large. For this reason we assume that there is a fixed bound T on the time allotted to the calibration procedure. We use a time bound rather than a bound on the number of simulator invocations because the value of some parameters can impact the simulator's space- and time-complexity.

B. Calibration Algorithm Implementations

Many algorithms can be used for solving the simulation calibration problem. These include simple searches, standard optimization algorithms such as gradient descent, genetic algorithms, or Machine Learning algorithms such as Bayesian optimization. Our goal in this work is not to determine which level of algorithm sophistication is sufficient. The answer to this question is likely highly dependent on the use case at hand (simulated system, simulator execution time, number of parameters to calibrate). Instead our primary goal is to determine if even simple optimization algorithms can improve upon the state of the art of manual simulator calibration. We consider the following such algorithms:

- **Grid Search** (GRID) – This algorithm evaluates all parameter combinations by subdividing the parameter space evenly in each parameter range. As the number of subdivisions is not known in advance, each time all current subdivisions of the range have been sampled, a new set of points to sample is determined using the mid-points between each pair of already sampled points. The initial ranges are the parameter value bounds provided by the simulator developer. Thus, given p parameters to calibrate, each parameter can take one of approximately $\sqrt[p]{N}$ (evenly spaced) values in its range, where N is the total number of simulation invocations completed before the time bound T has been reached.

- **Random search** (RANDOM) – This algorithm simply evaluates sets of random parameter values, where each value is sampled uniformly in its parameter range.

- **Gradient Descent –** This algorithm uses a random starting point in the parameter space. At each iteration the gradient is approximated by sampling points a distance δ away along each dimension. A standard backtracking line search is then used to compute the "learning rate," i.e., by how much to move along the gradient to determine the next point that should be sampled. When the change in the objective function between two iterations is less than ϵ, the current search path is terminated, and a new starting point is randomly selected. For all results in this paper we use $\delta = 0.0001$ and $\epsilon = 0.01$. For completeness, we did consider two variations of this algorithm:
  1) **Dynamic** (GDDYN) – At each iteration the value of δ is updated to be the learning rate determined by the backtracking line search;
  2) **Fixed** (GDFIX) – The value of δ remains constant regardless of the learning rate.

In all our experimental results these two variants lead to almost always identical simulation accuracy. Hence, in all that follows the results for GDDYN are omitted.

In all the above, random numbers are generated using a pseudo-random number generator seeded with the same seed. In our experiments all algorithms use the same bound $T = 6$ hours. Each algorithm executes one simulation on each core of a dedicated 2.5GHz Intel Xeon Gold 6248 40-core CPU.

IV. Case Study

A. Context and Objective

In this section we present a case study for an application in the field of High Energy Physics (HEP). Distributed computing platforms are used to support the high compute and storage demands of many HEP applications, for processing data generated by the Large Hadron Collider (LHC) experiments and simulations. Specifically, we consider the processing of data generated by LHC experiments conducted for the Compact Muon Solenoid (CMS) collaboration [31], which, in 2022, required ∼415PB of tape storage and more than 1.94 billion CPU-hours. The generated data, which describes particle collision events, can be split into chunks that can be processed and stored independently of each other. The processing of one event entails multiple data reduction/transformation steps until a final analysis step produces an output that can be stored on a single computer and analyzed to generate scientific results. This data processing workload is performed on a multi-site distributed computing platform, the Worldwide LHC Computing Grid (WLCG) [32], using various software infrastructures, such as HTCondor [33] for distributing computation and XRootD [34] for distributing storage.

Researchers need to estimate the execution times of current HEP workloads of interest on (subsets of) WLCG to plan experiments, to explore various hardware resource provisioning options, and to ensure that future CMS workloads can achieve acceptable performance. A key performance driver of scientific distributed computing applications is data locality, and HEP workloads are no exception. XRootD, which is deployed on WLCG, makes it possible to deploy data caches (called "proxy storage services") that can perform in-memory or on-disk caching. CMS researchers need to compare different cache deployment options in terms of the performance boost that caching can bring to current and future workloads. The main objective is thus to explore the large design space of combinations of hardware resource provisioning, cache deployment, and scheduling options, as well as workload configurations. Achieving this objective via real-world experiments would be too resource-consuming, especially since WLCG is used daily in production for running critical workloads. Also, real-world experiments cannot be used to explore hypothetical (future) scenarios. As a result, this objective can only be achieved by conducting simulation experiments.

B. Methodology

Simulator – We have developed a simulator [35] in C++ using the WRENCH and SimGrid simulation frameworks (see Section II-A). The simulator takes as input a description of a workload to execute and of the WLCG platform on which to execute it. A *workload* consists of a set of independent jobs, where each job consists in reading input files of given sizes, performing some volume of computation per byte of input, and writing an output file of a given size. The user can specify data and compute volumes either as constant values or as probability distributions from which values are sampled. A *hardware platform* consists of multiple sites interconnected over a wide-area network. One or more of these sites hosts a storage service that stores all initial input data for all jobs. Each site comprises multi-core compute nodes, each of which can use its local disk to cache input data. The simulator takes as input a number between 0 and 1, called the *ICD* (Initially Cached Data), that denotes the fraction of input files that are initially stored in these caches. These compute nodes are interconnected via a local network.

TABLE II
Hardware Platform Configuration Specifications.

Platform	RAM page cache	WAN interface
SCFN	disabled	10 Gbps
FCFN	enabled	10 Gbps
SCSN	disabled	1 Gbps
FCSN	enabled	1 Gbps

Ground Truth Data – Ground-truth data was obtained with a workload that comprises 48 jobs, where each job takes 20 files as input, each of size of ∼427MB. This workload was executed on WLCG using one compute site and a remote storage size, interconnected together via a wide-area network. The compute site hosts three compute nodes that are homogeneous, but two of these nodes have 12 cores while the third one has 24 cores. All three nodes host a local HDD cache, and are connected together via a local network. This platform configuration is depicted in Figure 1. The workload was executed for ICD values ranging from 0 to 1 in 0.1 increments.

Each execution of the workload on the platform was conducted for 4 different configurations of the hardware platform. Specifically, two different network interfaces can be used for the compute site to connect to the remote storage site (1 Gbps or 10 Gbps), and at each compute node the use of an in-RAM disk cache (the Linux Page Cache) can be enabled or disabled. These platform configurations are summarized in Table II (FC and SC stand for Fast Cache and Slow cache, respectively, and FN and SN stand for Fast Network and Slow Network, respectively). We treat each of these configurations as a different platform and perform simulation calibrations independently. Because they correspond to different ground truths with different hardware configurations, they cannot be used as a larger aggregated dataset that can be used for computing a single calibration. This is because our calibration parameters pertain to the platform hardware characteristics, as described hereafter.

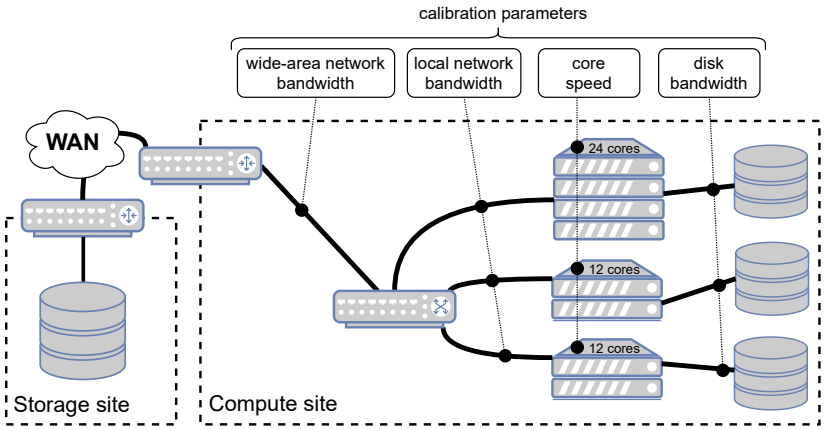

Fig. 1. Execution platform.

Calibration Parameters – In this case study the goal is to calibrate the following four simulation parameters, which are depicted also in Figure 1:

- Compute node core speed (flop/sec);
- Disk bandwidth (bit/sec);
- LAN bandwidth (bit/sec); and
- WAN bandwidth (bit/sec).

Although the simulator takes as input core speed floating point operations per second, these are better understood as work unit per second, where the work unit is application-specific. The simulator comes with two other parameters (which we consider in SectionIV-C4). The SimGrid and WRENCH frameworks provide simulation models configurable via hundreds of parameters (for which we use the default values these frameworks provide). As a result, our parameter search space has low dimension, which is why one can expect the simple algorithms in Section III-B to perform well.

Accuracy Metric – The domain scientists intending to use this simulator to achieve the objectives outlined in Section IV-A have identified performance metrics of interest. These are the the average job execution times for each of the 3 compute nodes at the compute site and for each of the 11 ICD values, for a total of 33 metrics. In this case study we consider a single aggregate metric to quantify simulation accuracy: the Mean Relative Error (MRE), in percentage, of these 33 metrics. A lower MRE value means a better accuracy.

Domain Scientist Calibration – In this case study we quantify the improvement that automated calibration can bring w.r.t. a calibration performed manually by a domain scientist. This manual calibration was performed by the second author of this paper. The approach was incremental. First, the core compute speeds were calibrated based on ground-truth data obtained from FCFN (to 1,970 Mflops), so as to minimize the overhead of network and I/O operation. Second, the external network bandwidth was calibrated for SCSN and FCSN (to 1.15 Gbps). For SCFN and FCFN, it was assumed that the same ratio of effective bandwidth to hardware-specified bandwidth applies, and thus the external network bandwidth was set to 11.5 Gbps (10x higher than for SCSN and FCSN). Third, the bandwidth of the HDD caches was calibrated based on SCFN (to 17 MBps). Some other parameter values were not calibrated, but their values were determined based on knowledge of the platform and simple benchmarks (and variations of these values were observed to have negligible impact on the simulation). These parameter values are: the internal network bandwidth (set to be 10 Gbps) and the Linux page cache speed (set to 1 GBps).

The manual calibration was done by starting from an initial guess (based on hardware specifications) and searching for a value in a neighborhood of this guess until a good match between the ground-truth data and the simulated data was observed. For the network bandwidths, the ground truth data exhibits low variance across job execution times, and there is negligible difference between the ground-truth data and the data obtained in simulation when using the calibrated parameters. For the HDD cache speed, there is higher variance across job execution times, especially at high ICD, due to more concurrent HDD reads to the cache. HDD effects (e.g., seek times) are not modeled by the simulator, and as a result the simulator does not produce the same variance. The calibration was performed to match the simulated data to the average of the ground-truth data. All details on this manual calibration procedure, which we denote as HUMAN, are available in [36].

Parameter Ranges – As explained in Section III-A, our automated calibration procedure requires that a range of possible values be specified for each parameter. In this case study, all parameters are given the same 2^{20} - 2^{36} range. This range was determined loosely based on reasonable expectations regarding hardware specifications, assuming that no specific knowledge of the platform is available.

C. Results

1) Simulation Accuracy: Table III shows MRE values for all calibration methods. The first observation is that the automated calibration methods almost always improve on HUMAN for all platforms. The exception is GRID for the SCFN platform, which leads to MRE higher than HUMAN by less

TABLE III
MRE FOR CALIBRATION METHODS AND PLATFORMS.

		Platform			
		SCFN	FCFN	SCSN	FCSN
Method	HUMAN	23.21%	274.20%	18.48%	196.24%
	RANDOM	22.07%	1.02%	14.69%	4.20%
	GRID	24.10%	3.08%	16.72%	8.48%
	GDFIX	22.90%	1.50%	15.83%	6.59%

than a point. The improvement is by only a few points for SCFN and SCSN, but more than 150 points for FCFN and FCSN. For these last two platform configurations recall that the manual calibration procedure simply assumes the value for the Linux page cache speed to be 1 GBps (see Section IV-B. This is likely the cause for the high MRE values, as the automated calibration methods compute values that are higher by $\sim$10x. The poorest performing algorithm is, expectedly, GRID. But we note that all our algorithms lead to similar MRE. This is because our search space is of low dimensionality and even simple algorithms, such as GRID and RANDOM, are able to find a good calibration. GDFIX is not significantly more effective because the objective function is "mostly flat" along several dimensions (for parameters that do not pertain to a bottleneck resource, as explained in the next section).

2) Bottleneck Resources: Although our algorithms lead to similar simulation accuracy, they actually compute quite different calibrations. Table IV shows calibrated parameter values computed by all calibration methods for platform SCSN. We observe that all methods compute very similar values for the disk bandwidth parameter (between 16 and 17 MBps). However, for some of the other parameters, the computed values can be wildly different. For instance, the values for the WAN bandwidth range from 0.27 Gbps to 57 Gbps, while the actual value is likely around 1Gbps. The reason is that the performance of the workload whose execution is simulated is driven by a single bottleneck resource. Parameter values pertaining to other resources thus have little impact on the simulated execution. In SCSN, the bottleneck for our ground-truth workload is the disk because the Linux page cache is disabled. The same observation can be made for the other three platforms, where all algorithms compute almost the same parameter value for the relevant bottleneck. Note that the HUMAN calibrations (for each platform) have values that are likely more accurate for non-bottleneck resource parameters due to the incremental manual calibration approach described in Section IV-B, which benefits from specific knowledge that the domain scientist has regarding platform configurations.

The problem with the above is that the computed calibrations are not generalizable to all application workloads. That is, the calibrated simulator is valid only to simulate the execution of workloads that would experience the same performance bottleneck as the ground-truth workload. Specifically, our calibrated simulator (using any of our algorithms), is only valid for simulating the execution of workloads with the same ratio of compute to data volumes as the ground-truth workload. For these workloads, the simulator is useful as it produces valid results for simulating configurations with more or fewer jobs, with more of fewer compute nodes and/or cores, and with different ICD values.

There are two solutions, which can be combined, for making calibrations generalizable to workloads with a range of ratios of compute to data volumes. The first solution would be the use of more ground-truth data obtained from the execution of workloads with different enough such ratios that they experience different bottlenecks when executed on the same platform. In this case study, however, the domain scientist collected ground-truth data for only one workload. The reasons are: (i) producing a calibrated simulator for this workload only is still useful, as explained above; and (ii) collecting ground-truth data is labor-, time, and energy-consuming. To allow for a fair comparison between the HUMAN calibration and our automated calibration approach, our calibration algorithms use that same ground-truth data. The second solution would entail defining and using a simulation accuracy metric whose value is not solely driven by parameter values that pertain to the bottleneck resource. The metric used in this case study (defined in Section IV-B) is an aggregate metric that does not capture the temporal structure of the workload's execution, but only takes into account average job execution times. As a result, the durations of activities (computations, I/O operations, network communications) that do not execute on the bottleneck resources (but execute concurrently with activities that do execute on these resources) have no impact on simulation accuracy. A metric that captures the duration of these activities would instead force the calibration algorithms to calibrate more than just the parameters that pertain to bottleneck resources. Such a metric could include, for instance, a measure of the discrepancy between the start and/or end times of all data transfers, I/O operations, and computations.

The questions of which level of diversity of ground-truth data is necessary and which accuracy metric is sufficient to ensure that automated calibration algorithms produce generalizable calibrations are beyond the scope of this paper and left for future work. Answering these questions will require conducting multiple case studies for different application domains and collecting new ground-truth data.

3) Using Less Ground-Truth Data: As seen in the previous section, more diverse ground-truth data is needed to obtain calibrations that are valid for the full spectrum of application workload configurations. Given our available ground-truth data in this case study, in all that follows we limit our scope to workloads that have the same compute-data ratio as that of the ground-truth workload. But within this scope, one may then wonder whether good calibrations can be computed automatically using less ground-truth data. Specifically, an interesting question is whether good calibrations can be computed based on only a subset of the ICD values. If using less ground-truth data is feasible, then the result is time, labor, and energy savings for the overall simulation calibration procedure.

To answer the above question we use one of our algorithms

TABLE IV
CALIBRATED PARAMETER VALUES FOR PLATFORM SCSN.

	Core speed	Disk bandwidth	LAN bandwidth	WAN bandwidth
HUMAN	1,970 Mflops	17 MBps	10.0 Gbps	1.15 Gbps
RANDOM	823 Mflops	17 MBps	6.1 Gbps	21.0 Gbps
GRID	1,073 Mflops	17 MBps	17.0 Gbps	0.27 Gbps
GDFIX	778 Mflops	16 MBps	2.5 Gbps	57.0 Gbps

TABLE V
BEST, MEDIAN, AND WORST MRE WHEN CALIBRATING USING SUBSETS OF THE ICD VALUES USING GDFIX FOR PLATFORM FCSN.

# ICD values	# Subsets	Best	Median	Worst
1	5	20.51%	52.00%	7008.44%
2	10	4.20%	5.52%	21.10%
3	10	4.20%	4.20%	10.02%
11	1	6.59%	6.59%	6.59%

(GDFIX) to compute calibrations when using subsets of a 5-element set of ICD values $\{0.0, 0.3, 0.5, 0.7, 1.0\}$. Table V shows results for calibrating using all five 1-element subsets, all ten 2-element subsets, and all ten 3-element subsets. The last row of the table shows results when using all 11 ICD values for calibration, as done in the previous section. These results are for platform FCSN (results are similar for other platforms). We do not show results for each individual subset, but show instead the best, median, and worst MRE values over all subsets with the same cardinality.

When calibrating using one ICD value the MRE range is 20.51%-7,008.44%, with the worst MREs (higher than 5,000%) obtained when calibrating based on one of the two extreme ICD values 0.0 and 1.0. This is expected as with these values the caching behavior is markedly different than that with intermediate values. Calibrating using only one of these intermediate values leads to MRE between 20.5% and 52.0%, with the lowest MRE values achieved when calibrating based on ICD 0.5. When using two ICD values the MRE improves drastically. Although the worst MRE is at 21.10%, the median is at 5.52%. In fact, only one of the ten possible subsets leads to MRE above 9%, for subset $\{0, 0.3\}$. When using three ICD values, the worst MRE improves to 10.02%. Here again, the median is equal to the best. Only subset $\{0, 0.3, 0.5\}$ leads to MRE above 7%. In these results, when using 2- or 3-element subsets, the worst performing subset is always the one that includes only the smallest ICD value. As long as there is reasonable diversity in ICD values, e.g., some below 0.5 and some above 0.5, the automated calibration based on two or three ICD values leads to accuracy on par with that obtained when calibrating with all 11 ICD values.

Calibrating using n ICD values can lead to better accuracy than calibrating with $n' > n$ ICD values, i.e., using less ground-truth data. For instance, the best MRE value when using all 5 ICD values is 6.59%, but is 4.20% when using 2 or 3 ICD values. The main reason is that the same amount of time T is allotted to the calibration procedure regardless of how many ICD values are used. Evaluating the accuracy of a calibration requires n'/n fewer simulator invocations when using n ICD values as opposed to n' ICD values. Thus using fewer ICD values makes it possible to explore the parameter space more thoroughly within time T.

We conclude that as long as a reasonably diverse set of ground-truth data is used, good calibrations can be computed even when using relatively few values. Nevertheless, in the following sections results are presented for calibrations computed using all 11 ICD values.

4) Trade-off between Speed and Accuracy: Our simulator takes as input more parameters than the four that we have calibrated in previous sections. We now consider two specific additional parameters. The first parameter is the XRootD block size, B. Each file in XRootD, like in most storage systems, is partitioned into blocks. The jobs in the workload process input files block by block, so that reading and processing data is done in a pipelined fashion. The second parameter is the buffer size, b, which specifies the internal buffer size used by a storage service, for the purpose of pipelining I/O and network operations, as done in production storage systems.

The B and b parameters correspond to software configuration parameters in the real-world system. Picking realistic values for them could be done by inspecting the system's configuration files. This was not done for the WLCG platform for this case study but, regardless, values are likely a few MBs or on the order of KBs (e.g., the default XRootD block size is 2MB). These parameters drive the number of discrete events that must be simulated. Given a job in the workload that needs to process s bytes of data, the number of simulated events for this job's execution is $O(s/B + s/b))$. If B and/or b are low relative to s, the simulation time can become prohibitively high. Hence, for these two parameters the goal is not to find values that are as realistic as possible. Instead, the goal is to set the values of these parameters so that the simulation time is below some user-defined threshold and then calibrate all other parameters automatically. The question is whether this calibration can still lead to good simulation accuracy. In other words, can the automated calibration of the other parameters compensate for the potential loss of accuracy due to simulating the execution at a higher granularity (i.e., larger block and buffer sizes) than the real-world system?

To answer this question we consider four combinations of

B and b values, so that the average simulation time is $\sim$1 sec ($B = 10^{10}$ byte, $b = 10^8$ bytes), $\sim$3 sec ($B = 10^9$ bytes, $b = 10^7$ bytes), $\sim$30 sec ($B = 10^8$ bytes, $b = 10^6$ bytes), or $\sim$5 min ($B = 10^7$ bytes, $b = 10^5$ bytes). We then run our automated calibration procedure for platform FCSN for each of our algorithms. In all other sections we use $B = 10^8$ bytes and $b = 10^6$ bytes (for $\sim$ 30-sec simulation times).

TABLE VI
MRE VS. AVERAGE SIMULATION TIME FOR PLATFORM FCSN.

Sim. time	GDFIX	GRID	RANDOM
$\sim$1 sec	3.13%	4.50%	2.93%
$\sim$3 sec	4.26%	10.85%	3.26%
$\sim$30 sec	6.59%	8.48%	4.20%
$\sim$5 min	13.58%	28.33%	4.02%

Table VI shows MRE vs. average simulation time for our three algorithms. In general we observe that MRE increases as the simulation time increases. But there are some exceptions: GRID leads to higher MRE with 3-sec simulation times than with 30-sec simulation times; and RANDOM leads to higher MRE with 30-sec simulation times than with 5-min simulation times. This may seem surprising given that all simulation parameter values that are sampled when using the longer simulation time are also sampled when using the shorter simulation time. However, simulation parameters that lead to low MRE for particular B and b values may lead to high MRE for different B and b values. For instance, the best found calibration that achieves an MRE of 8.48% with $B = 10^8$ and $b = 10^6$ (30-sec simulation time), leads to an MRE of 30.42% when used with $B = 10^9$ and $b = 10^7$ (3-sec simulation time).

Regardless, the key observation is that for all algorithms the best MRE is achieved for the fastest simulation time, i.e., for the largest B and b values With larger B and b values the simulation has a much higher granularity than the real-world system, which would seem to imply worse accuracy. However, with these large values the simulation time is short, meaning that the calibration procedure can better explore the parameter space, allowing it to find parameter values that lead to better accuracy in spite of the higher granularity. A lower granularity in the real-world system means better utilization of the hardware resources due to finer-grain pipelining of I/O, network, and compute activities. Pipelining thus increases the effective speed of I/O, network, and compute resources for each job. In simulation this same increase can be achieved instead by using a higher granularity and at the same time increasing the speed of the corresponding simulated hardware resources, at least within some bounds. In this case study, doing so allows the user to obtain a calibrated simulator that is both fast and accurate. Because our calibration procedure is automated, it would be straightforward for users of the simulator to explore the accuracy-speed design space to achieve whatever user-specific trade-off is the most desirable. Making this determination manually would be prohibitively labor-intensive.

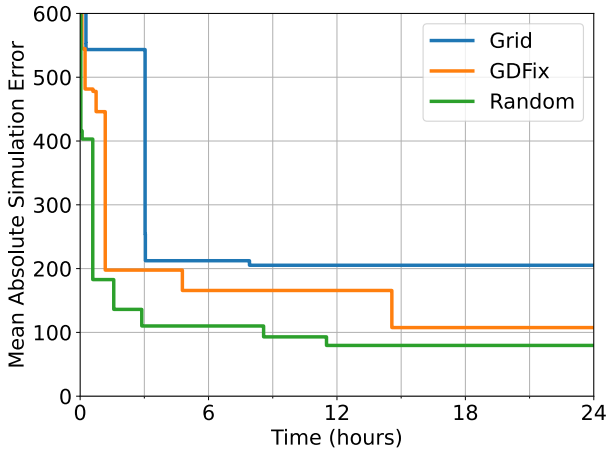

Fig. 2. Absolute simulation error vs. time for platform FCSN.

5) Impact of the Time Bound T: All previous results were obtained for an arbitrarily fixed calibration time bound T of 6 hours. Figure 2 plots absolute simulation error vs. time, for time up to 24 hours, for platform FCSN (results are similar across all platforms). These results are obtained using $B = 10^8$ and $b = 10^6$ bytes for the XRootD block size and buffer size parameters, for which the simulation time is $\sim$30 sec.

As expected, all curves are non-increasing, with a very sharp initial decrease. GRID leads to the worst results with the slowest convergence. RANDOM converges the most rapidly and leads to the lowest error overall. GDFIX is in between. Some improvements can be achieved by setting $T > 6$ hours, especially for RANDOM and GDFIX, with both algorithms converging to similar error values. Using a shorter T $\sim$3 hours would have produced only marginally higher errors than with $T = 6$ hours.

V. CONCLUSION

We have shown that the state of the art of PDC simulator calibration comprises undocumented and/or labor-intensive ad-hoc approaches, which motivates for developing automated calibration methods. Via a case study from the field of High Energy Physics, we have demonstrated that even simple algorithms can be used to improve upon a calibration computed by a domain scientist. We have also shown that this can be achieved with reduced amounts of ground-truth data, and that automated calibration makes it possible to achieve desirable trade-offs between simulation speed and simulation accuracy.

A clear future direction is to augment our case study and to perform case studies for other PDC systems and simulators thereof. Doing so will allow investigating which amount and diversity of ground-truth data, and which accuracy metric definitions, are sufficient to compute calibrations that are robust: the calibrated simulator should yield accurate results for the full spectrum of possible application workload configurations. In our case study in this work we have kept the calibration parameter space at only 4 dimensions. But in practice, for

this and other case studies, it could be much larger with with hundreds of parameters. The simple algorithms we have considered in this work will likely no longer be effective, and another clear future direction is the use of Machine Learning algorithms. In particular, Bayesian Optimization is an attractive proposition as it is highly effective for optimizing black-box functions that are relatively expensive to evaluate, such as simulation accuracy metrics whose evaluation entails invoking a simulator.

Acknowledgments

This work was partially supported by NSF Awards #2106059 and #2103489, the German Federal Ministry of Education and Research (project FIDIUM 05H21VKRC2) and the Institute of Experimental Particle Physics (ETP) at the Karlsruhe Institute of Technology, Germany. The technical support and advanced computing resources from University of Hawai'i Information Technology Services - Cyberinstructure, funded in part by the NSF MRI Award #1920304, are gratefully acknowledged.

References

[1] R. Buyya and M. Murshed, "GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing," *Concurrency and Computation: Practice and Experience*, vol. 14, no. 13-15, pp. 1175–1220, Dec. 2002.

[2] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, Jan. 2011.

[3] S. Ostermann, K. Plankensteiner, R. Prodan, and T. Fahringer, "GroudSim: An Event-Based Simulation Framework for Computational Grids and Clouds," in *Euro-Par 2010 Parallel Processing Workshops*, M. R. Guarracino, F. Vivien, J. L. Träff, M. Cannatoro, M. Danelutto, A. Hast, F. Perla, A. Knüpfer, B. Di Martino, and M. Alexander, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 305–313.

[4] G. Kecskemeti, "DISSECT-CF: A Simulator to Foster Energy-Aware Scheduling in Infrastructure Clouds," *Simulation Modelling Practice and Theory*, vol. 58, pp. 188–218, 2015.

[5] J. Cope, N. Liu, S. Lang, P. Carns, C. Carothers, and R. Ross, "CODES: Enabling Co-Design of Multilayer Exascale Storage Architectures," in *Proc. of the Workshop on Emerging Supercomputing Technologies*, 2011.

[6] M.-Y. Hsieh, R. Riesen, K. Thompson, W. Song, and A. Rodrigues, "SST: A Scalable Parallel Framework for Architecture-Level Performance, Power, Area and Thermal Simulation," *The Computer Journal*, vol. 55, no. 2, pp. 181–191, 2012.

[7] E. U. Yousuf Khan, T. Rahim Soomro, and M. Nawaz Brohi, "iFogSim: A Tool for Simulating Cloud and Fog Applications," in *Proceedings of the International Conference on Cyber Resilience*, 2022, pp. 01–05.

[8] H. Casanova, R. Ferreira da Silva, R. Tanaka, S. Pandey, G. Jethwani, W. Koch, S. Albrecht, J. Oeth, and F. Suter, "Developing Accurate and Scalable Simulators of Production Workflow Management Systems with WRENCH," *Future Generation Computer Systems*, vol. 112, pp. 162–175, 2020.

[9] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, "Versatile, Scalable, and Accurate Simulation of Distributed Applications and Platforms," *Journal of Parallel and Distributed Computing*, vol. 75, no. 10, pp. 2899–2917, 2014.

[10] 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*, vol. 23, no. 4, 2013.

[11] "The WRENCH Project," http://wrench-project.org/, 2022.

[12] "The SimGrid Project," http://simgrid.org/, 2022.

[13] 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.

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

[15] 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 International Symposium on Cluster Computing and the Grid*, 2015.

[16] 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*, vol. 18, no. 8, pp. 2387–2400, 2017.

[17] A. Rizvi, T. Toha, M. Lunar, M. Adnan, and A. Alim Al Islam, "Cooling Energy Integration in SimGrid," in *Proc. of the 2017 International Conference on Networking, Systems and Security (NSysS)*, 2017, pp. 132–137.

[18] 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 *Proc. of 2017 IEEE International Conference on Cluster Computing*, 2017, pp. 92–102.

[19] 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 *Proc. of the 2015 IEEE 21st International Conference on Parallel and Distributed Systems*, 2015, pp. 481–490.

[20] L. Stanisic, "A Reproducible Research Methodology for Designing and Conducting Faithful Simulations of Dynamic HPC Applications," Ph.D. dissertation, Université Grenoble Alpes, France, 2015.

[21] T. Cornebize, A. Legrand, and F. C. Heinrich, "Fast and Faithful Performance Prediction of MPI Applications: the HPL Case Study," in *Proc. of the 2019 IEEE International Conference on Cluster Computing*, 2019, pp. 1–11.

[22] T. Andel and A. Yasinsac, "On the Credibility of MANET Simulations," *Computer*, vol. 39, no. 7, pp. 48–54, 2006.

[23] G. Flores, M. Paredes-Farrera, E. Jammeh, M. Fleury, and M. Reed, "OPNET Modeler and Ns-2: Comparing the Accuracy of Network Simulators for Packet-Level Analysis Using a Network Testbed," *WSEAS Transactions on Computers*, vol. 2, no. 3, 2003.

[24] P. Garrido, M. Malumbres, and C. Calafate, "Ns-2 vs. OPNET: A Comparative Study of the IEEE 802.11e Technology on MANET Environments," in *Proc. of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, 2008.

[25] J. Lessmann, P. Janacik, L. Lachev, and D. Orfanus, "Comparative Study of Wireless Network Simulators," in *Proc of the 7th International Conference on Networking*, 2008, pp. 517–523.

[26] P. Hurni and T. Braun, "Calibrating Wireless Sensor Network Simulation Models with Real-World Experiments," in *Proc. of the 8th International IFIP-TC 6 Networking Conference*, ser. Lecture Notes in Computer Science, vol. 5550. Springer, 2009, pp. 1–13.

[27] M. Hofmann, "On the Complexity of Parameter Calibration in Simulation Models," *The Journal of Defense Modeling and Simulation*, vol. 2, no. 4, pp. 217–226, 2005.

[28] Y. Liu, O. Batelaan, F. Smedt, J. Poorova, and L. Velcicka, "Automated Calibration Applied to a GIS-based Flood Simulation Model Using PEST," *Floods, from Defence to Management*, pp. 317–326, 01 2005.

[29] T. Yang, Y. Pan, J. Mao, Y. Wang, and Z. Huang, "An Automated Optimization Method for Calibrating Building Energy Simulation Models with Measured Data: Orientation and a Case Study," *Applied Energy*, vol. 179, 2016.

[30] J. Hourdakis, P. G. Michalopoulos, and J. Kottommannil, "Practical Procedure for Calibrating Microscopic Traffic Simulation Models," *Transportation research record*, vol. 1852, no. 1, pp. 130–139, 2003.

[31] The CMS Collaboration, "The CMS experiment at the CERN LHC," *Journal of Instrumentation*, vol. 3, no. 08, p. S08004, 2008.

[32] "The Worldwide LHC Computing Grid," https://wlcg.web.cern.ch/, 2023.

[33] "The HTCondor Software Suite," https://htcondor.org, 2023.

[34] "The XRootD Project," https://xrootd.slac.stanford.edu/, 2023.

[35] "Simulator implementation," https://zenodo.org/records/8300961, 2023.

[36] M. Horzela, "Measurement of Triple-Differential Z+Jet Cross-Sections with the CMS Detector at 13 TeV and Modelling of Large-Scale Distributed Computing Systems," Ph.D. dissertation, Karlsruhe Institute of Technology, 2023. [Online]. Available: https://doi.org/10.5445/IR/1000165566