OpenDC 2.0: Convenient Modeling and Simulation of Emerging Technologies in Cloud Datacenters

Fabian Mastenbroek[1,2], Georgios Andreadis[3,1,2], Soufiane Jounaid[2], Wenchen Lai[2], Jacob Burley[2], Jaro Bosch[2], Erwin van Eyk[2], Laurens Versluis[2], Vincent van Beek[3,2], and Alexandru Iosup[2,1]
[1]TU Delft, [2]VU Amsterdam, [3]Solvinity, the Netherlands – Contact: F.Mastenbroek@atlarge-research.com

***Abstract*—Cloud datacenters are important for the digital society, serving stakeholders across industry, government, and academia. Simulation is a critical part of exploring datacenter technologies, enabling scalable experimentation with millions of jobs and hundreds of thousands of machines, and what-if analysis in a matter of minutes to hours. Although the community has already developed powerful simulators, emerging technologies and applications in modern datacenters require new approaches. Addressing this requirement, in this work we propose OpenDC, a new platform for datacenter simulation. OpenDC includes novel models for emerging cloud-datacenter technologies and applications, such as serverless computing with FaaS deployment and TensorFlow-based machine learning. Our design also focuses on convenience, with a web-based interface for interactive experimentation, support for experiment automation, a library of prefabs for constructing and sharing datacenter designs, and support for diverse input formats and output metrics. We implement, validate, and open-source OpenDC 2.0, a significant redesign and release after a multi-year research and development process. We demonstrate the benefits of OpenDC for the field through a set of representative use-cases: serverless, machine learning, procurement of HPC-as-a-Service infrastructure, educational practices, and reproducibility studies. Overall, OpenDC helps understand how datacenters work, design datacenter infrastructure, and train the next generation of experts.**

***Index Terms*—OpenDC, datacenter, simulation, modeling, use-cases, experimentation, performance analysis.**

I. Introduction

Cloud infrastructure has become a cornerstone of today's digital society [1]. Stakeholders across industry, government, and academia employ diverse cloud services hosted by datacenters, and expect services to be reliable, high speed, and low cost. Faced with growing demand, datacenter architects must address complex challenges in distributed systems [2], [3], software engineering [4], and performance engineering [5]. The consequences of bad decisions can be financial penalties or even loss of customers, so conservative datacenter operators demand new technologies be well-tested [6]. *Simulation* is essential in enabling large-scale and fine-grained exploration, analysis, and comparison of datacenter technologies. Although the community has already built many high-quality simulators [7]–[9], the many technologies and applications emerging in cloud datacenters, and the need to educate an increasing number of professionals about them, raise new challenges—in creating new simulation models and in engineering new simulation platforms. Addressing these challenges, we propose in this work OpenDC, an open-source datacenter simulation platform, and present five representative use-cases to demonstrate the value of OpenDC for the field.

As in other fields of science and technology, simulation is critical for the development and adoption of new techniques in datacenter operations. Simulators such as Grid/CloudSim [7], SimGrid [8], and iCanCloud [9] have demonstrated the ability of simulators to represent and understand the operation of very complex operations, at cluster- and datacenter-level. In contrast, the main alternatives to simulation, such as mathematical analysis [10], [11] and real-world experimentation [12], do not yet seem viable at the scale, dynamicity, and diversity required by cloud datacenters. In particular, conducting experiments on physical infrastructure at datacenter-scale is prohibitively expensive, time consuming, and difficult to reproduce [13], notwithstanding environmental concerns (see Section II-C).

Current simulators, albeit much used in our community, still pose a variety of challenges. First, they need to *support frequent and substantial innovation* in cloud datacenters, and specifically to provide new models that capture emerging technologies and applications, such as serverless computing [14] and machine learning workloads running in datacenters [15]; Section II-B describes a general model. Second, we observe the difficulty of using simulators as *platforms for research communities*: we need to improve the tools for interaction and visualization, the support offered for diverse input formats and output metrics, the process of designing and sharing (parts of) complex datacenters, etc. Third, simulators can help *different stakeholders*—from experts doing analysis and design, to managers taking essential decisions, to students learning the basics of the field—, raising the challenge of engineering such software for more general users.

Addressing these challenges, we propose in this work OpenDC 2.0, an open-source platform for modeling, simulation, and experimentation with cloud datacenters. OpenDC is not meant to replace all other simulators currently in use by the community. Instead, it aims to complement the current state-of-the-art, in particular through its focus on use-cases that have not been systematically supported before, on convenience as a platform for the research community, and on software engineering. Overall, our contribution is three-fold:

1) We design OpenDC 2.0, an advanced platform for datacenter simulation (Section III). OpenDC is the first simulator to integrate serverless and machine learning execution, both emerging services already offered by all major cloud providers. OpenDC also models all the major operational layers of typical clouds, from datacenter infrastructure and virtualization, to resource management and scheduling. OpenDC provides convenience in the de-

sign, understanding, and exploration of cloud datacenters through discrete-event simulation.

2) We present five representative use-cases for datacenter simulation using OpenDC (Section IV). These use-cases explore a variety of important and emerging topics in the community: serverless, machine learning, procurement of HPC-as-a-Service infrastructure, reproducibility, and educational practices. Using OpenDC, we gain new insights, e.g., whether the new cost policies of AWS Lambda benefit the average user, or whether datacenters adding the HPC workloads of small and medium enterprises should procure vertically or horizontally scaled infrastructure.
3) We open-source OpenDC 2.0. Our work on the OpenDC simulator started in 2016 [16]. About 4 years after version 1.0, we release an innovative new version. Our development follows modern engineering practices, from using co-routines for efficient simulation, to extensive testing and documentation. The complete material is online: https://github.com/atlarge-research/opendc

II. A Primer on Datacenter Simulation

OpenDC employs simulation for datacenter exploration. In this section, we explain what simulation is, model datacenter operation, and compare simulation to other approaches.

A. What is Simulation?

Simulation is the "imitation of a real-world process or system over time, enabling the study of, and experimentation with the internal interactions of complex systems" [17]. It is widely applied to complex systems in many domains of science and industry, including computer systems [7]–[9].

In this work, we consider only a particular method of simulation, *discrete-event simulation* [17], where the operation of a system is represented as a sequence of events over time, with the assumption that no changes occur in-between events. This allows direct progression between events, in contrast to continuous models. Almost all efforts to model cloud and datacenter operations employ discrete-event simulation, due to the sheer scale and complexity of datacenters and long-running nature of experiments; OpenDC does the same.

B. A General Model for Datacenter Simulation

We assume in this work the general model of datacenters and their operation based on [18] and depicted by Figure 1.

Workload: The workload consists of applications running on *physical machines*, *virtual machines (VMs)*, or *containers*. We consider also *app managers*, such as the big data framework Apache Spark, the machine learning framework TensorFlow, and the serverless framework OpenFaaS, which orchestrate virtualized workflows and dataflows for their users.

Our model also considers scientific workloads deployed on virtualized environments. These workloads are primarily comprised of conveniently (embarrassingly) parallel tasks—e.g., Monte Carlo simulations—forming *batch bags-of-tasks*.

Datacenter Resources: Workloads run on physical datacenter infrastructure. We model datacenter infrastructure as a set of physical clusters of possibly *heterogeneous hosts*, each host a node in a datacenter rack. A host can execute multiple VM- or container-workloads, managed by a *hypervisor*.

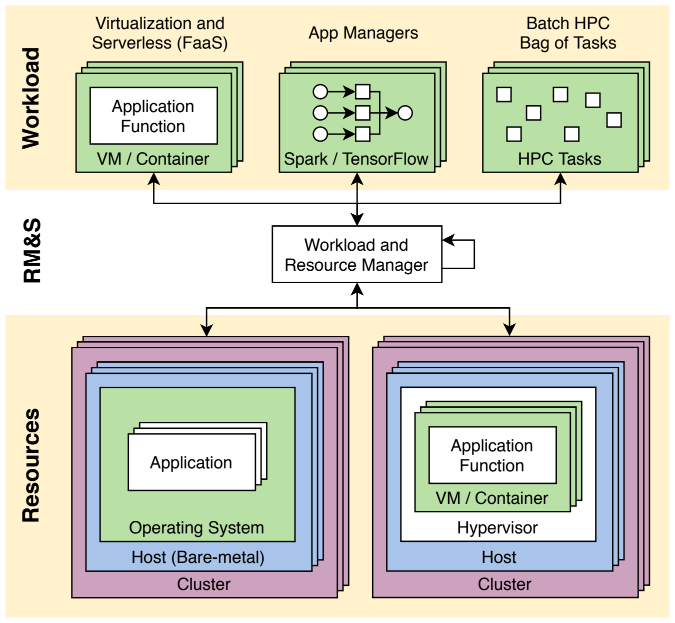

Figure 1: Generic model for datacenter operation.

We model in this work resource consumption of applications (e.g., CPU usage) per discretized time slices. Workloads report at each time slice their resource consumption to the hypervisor, which consolidates the requests and distributes the resources based on some scheduling policy: CPU resources are allocated between the workloads that request it, through *time-sharing* (if on the same cores) or *space-sharing* (if on different cores). We assume a generic memory model, with memory allocation constant over the runtime of a machine. As is currently common in industry, we allow overcommission of CPU resources [19], but not of memory resources [20].

Operational Phenomena: Cloud datacenters are complex hardware and software ecosystems, in which complex phenomena emerge [21]. Given the absence of a general model, we consider two very common operational phenomena: (i) *performance variability* caused by performance interference between collocated VMs [22], [23], modeled using a CPU-contention predictor for demanding business-critical workloads [22], and (ii) *correlated cluster failures*, based on a common model for space-correlated failures [24], where a failure may trigger more failures within a short time span, which together form a group. Currently, we model only *full-stop failures*: machines crash fully, with subsequent recovery after some duration.

Resource Management and Scheduling: We model a workload and resource manager that performs management and control of all clusters and hosts, and is responsible for the lifecycle of submitted workloads, including their placement onto the available resources [18]. The resource manager is configurable and supports various *policies* to distribute workloads over resources.

C. On the Benefits and Drawbacks of Simulation

Simulation might not always be the most appropriate tool. In this section, we compare it with other approaches.

Compared to Mathematical Analysis: Analytic models provide a fast and high-level mathematical approach for predicting performance, but it is difficult to capture in these high-level models the heterogeneity and complex interplay (e.g., the operational phenomena in Section II-B) of the hardware and software ecosystems present in datacenters. Accuracy is also only as good as the calibration data, which remain scarce.

Simulation, in contrast to purely analytical models, allows composing various models according to well-understood, system-level rules.

Compared to Real-world Experimentation: Although experiments on physical infrastructure deliver results closest to real-world operating conditions, they are difficult to reproduce due to various phenomena (e.g., [25]) affecting system performance and in turn measurements non-trivially.

Real-world experiments are time-consuming, expensive, and have a significant environmental impact: recently, we evaluated[1] over 6,000 scenarios running a month-long workload trace, which to simulate took 112 CPU hours on a modern machine, but would have taken 8.6 billion CPU hours to replicate on physical infrastructure, incurring an energy bill roughly equivalent to the annual energy consumption of the Netherlands (120 billion kWh).

Simulation, in contrast to real-world experimentation, also enables the possibility to investigate "what-if" scenarios. For example, an organization could want to try out the potential limitations of a large-scale architecture or protocol, and a datacenter may want to explore topological alternatives, without having to fully implement and deploy it.

III. The Design of OpenDC 2.0

In this section, we synthesize requirements and design around them a datacenter simulator, OpenDC 2.0.

Novelty: OpenDC 2.0 is a radical re-design and re-engineering effort over version 1 (2017). We contrast it to the current state-of-the-art in datacenter simulators, in Section V.

A. Timeline of the OpenDC project

We describe in this section the evolution of the OpenDC project since its inception, which we summarize in Figure 2. The project started mid-2016 with the aim to support the exploration of various datacenter concepts and technologies, and to enable new education practices and topics, by developing new scientific methods. In 2017, we started the development of OpenDC 1.0 and outlined our vision for the project [16].

The initial version of OpenDC, while suitable as prototype, suffered from issues such as scalability, maintainability, and extensibility, which made it difficult to deploy in practice. Addressing these issues, we started a significant redesign of the simulator early 2018, which has since formed the foundation of numerous projects within our research group.

[1]M.Sc. thesis by Georgios Andreadis, 2020. bit.ly/CapelinThesis

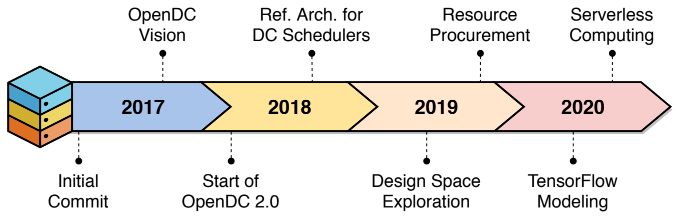

Figure 2: Timeline of the OpenDC project.

New, OpenDC 2.0 supports (1) the reference architecture for datacenter schedulers [18], and TensorFlow and serverless workloads, (2) systematic design-space exploration through experiment automation, and (3) procuring cloud infrastructure.

B. Requirements Analysis

We now synthesize the requirements addressed by OpenDC:

(R1) **Model cloud datacenter environments**. The system must enable the user to model cloud environments, within the scope introduced in Section II-B, e.g., with support for diverse resources, workloads, and policies.

(R2) **Support visual and interactive exploration**. Datacenter technology must be accessible to diverse users. The system must enable users to explore datacenter technologies in visually and interactively.

(R3) **Support the scale of modern cloud infrastructure**. Cloud infrastructure currently operates at an unprecedented scale. The system should operate efficiently to support large-scale cloud environments and workloads.

(R4) **Support relevant emerging scenarios** in the community. The system must model at least one class of the workloads and operations in:

a) **Serverless computing**, which is becoming widely adopted. In serverless computing, the cloud provider orchestrates the infrastructure for applications instead of the developer, and bills with fine-granularity [14].

b) **Machine learning (ML)**, which has gained much attention, enabling applications in various domains [26]. The training part of ML is particularly compute- and data-intensive, and thus has high energy footprint.

(R5) **Ensure interoperability with the existing ecosystem**. The community already employs various formats and tools to conduct research. To foster adoption, the system should be interoperable with existing tools.

(R6) **Enable re-use and sharing of designs**. The design of datacenters is a complex activity and demands technical expertise. To simplify the design process and decrease barriers to entry, the system should facilitate the re-use and sharing of designs, either by parts or entirely.

(R7) **Support educational purposes**. There is a lack of skilled human resources in the field [27]. The system should support various kinds of learning for diverse students.

(R8) **Adhere to modern software development standards**. The system should not only be useful for this work, but should evolve and adapt to future work. To this end, the system should be engineered to professional, modern software development standards.

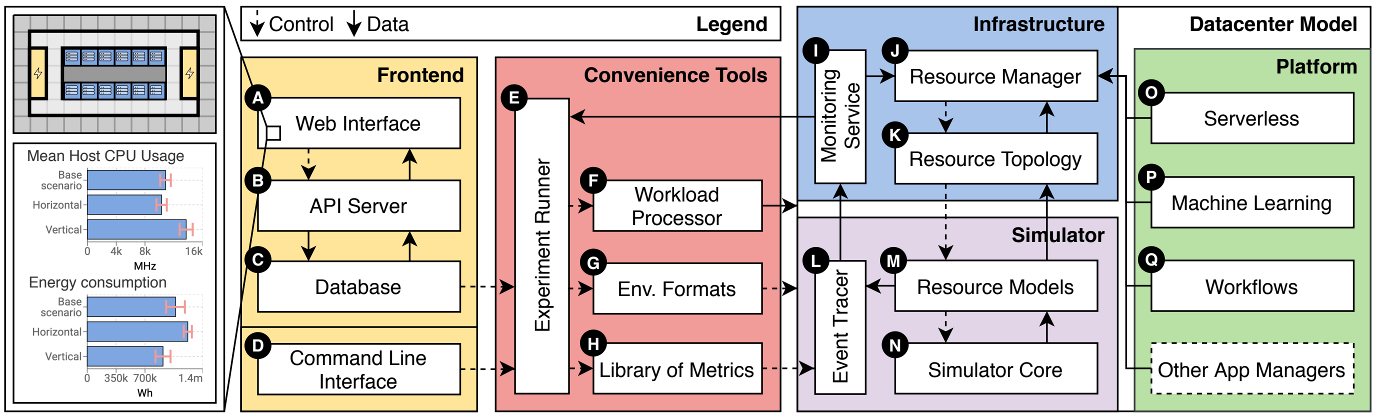

Figure 3: An overview of the architecture of OpenDC 2.0.

C. *Overview of the OpenDC Architecture*

We discuss in this section the high-level architecture of OpenDC, which we have depicted in Figure 3. At the highest level, the OpenDC architecture is composed of three main components: (i) a web and textual frontend, (ii) a model-driven discrete-event simulator, and (iii) a set of tools to assist with simulation. In turn, the datacenter model is a layered architecture representing the various abstraction levels offered by clouds, e.g., Infrastructure as a Service (IaaS), Platform as a Service (PaaS), building upon a small simulator core. We now discuss, in turn, each layer and sub-layer:

Frontend (addresses requirements R2, R7): In Figure 3, the *Web Interface* (Ⓐ) serves as the user-portal, through which stakeholders can interactively construct, share, and re-use datacenter designs (see Section III-D). With these designs, users can configure and conduct experiments. At any time, users can explore the automated plots and visual summaries generated by OpenDC, from single setups to comparative experiments. The *API Server* (Ⓑ) responds to web requests, acting as intermediary and business-logic between the web frontend on the one side, and database and simulator on the other side. The *Database* (Ⓒ) manages the state of the simulation platform, including topology models, historical data, simulation configurations, and simulation results.

Users may also deploy the simulator as a standalone package and utilize its *Command Line Interface* (Ⓓ). Although this does not provide the same usability and accessibility as the web interface, it is useful for conducting experiments in headless environments and simplifies reproducibility efforts.

Simulator (R1, R3): The foundation of OpenDC is the simulator. The *Simulator Core* (Ⓝ) provides a small set of primitives to enable simulated components using discrete-event simulation. The *Resource Models* (Ⓜ) model generic resource-sharing semantics, which are used to represent the behavior of datacenter resources and their scheduling policies. The *Event Tracer* (Ⓛ) traces events in the system, enabling in-depth monitoring of components. The simulator coordinates with cloud-level operational models, IaaS and PaaS, which we describe in the following, in turn.

Infrastructure (R1): The *Resource Manager* (Ⓙ) component models a typical IaaS such as AWS EC2, from where users can lease compute resources on-demand. Internally, the *Resource Topology* (Ⓚ) represents the resources available in the simulated datacenter, ranging from physical cluster nodes to VMs to containers (see Section II-B). The *Monitoring Service* (Ⓘ) is responsible for monitoring the datacenter resources.

Platform (R4): This level supports the execution of many platform-level operations, programmatically. This enables a variety of application types, as described in Section II-B. For example, the *Serverless* (Ⓞ) component models a *Function-as-a-Service* (FaaS) platform such as AWS Lambda (see also Section III-E). The *Machine Learning* (Ⓟ) component models the TensorFlow machine learning framework in the cloud (see also Section III-F). The *Workflows* (Ⓠ) component implements a generic workflow engine.

Convenience Tools (R5, R6): Supporting the simulation process is a collection of tools. The *Experiment Runner* (Ⓔ) automates the orchestration of experiments using OpenDC, enabling users to write declarative experiment specifications, and perform automated experiment design and optimization (e.g., evolutionary optimization). The *Workload Processor* (Ⓕ) enables reading, writing, and processing of workload traces in many of the formats used by the community. Currently, our tooling supports parsing, processing, and converting workload traces from the Grid Workload Archive [28], Workflow Trace Archive [29], and Parallel Workloads Archive [30], and several internal and ad-hoc formats. The *Environment Processor* (Ⓖ) is responsible for processing the datacenter designs of users and adds mechanisms to assemble, provision, and configure the simulated infrastructure. The *Library of Metrics* (Ⓗ) consists of re-usable collectors for a vast and diverse set of metrics utilized by the community, ranging from resource-level metrics (such as CPU usage or energy consumption) to application-level metrics (such as workflow makespan).

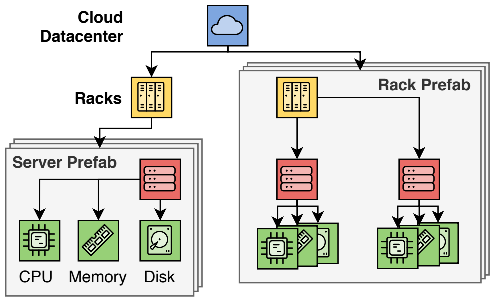

Figure 4: An abstract overview of prefabs.

D. *Prefabs*

The design of datacenters is a highly complex process. Faced with a shortage of datacenter designers, it is important to make datacenter design accessible to a wider range of people. Currently, to create simulations, users of simulators in the field need to specify each individual component used to build out the entire datacenter. This approach is time-consuming, error-prone, and requires a high level of technical expertise.

Addressing R6, we envision that prefabricated components (*prefabs*), that is, complete collections of components, could be "dragged and dropped" into a datacenter design. OpenDC implements prefabs, which accelerates prototyping. Our approach also allows for exporting of existing designs into prefabs, which allows users to quickly clone existing components to scale up their designs by resource-replication, replicate and re-use entire designs, and share datacenter designs for new or underrepresented workloads with the community.

Figure 4 depicts how components can be grouped together into prefabs, which are treated as any other objects that can be used in designing datacenters. Without prefabs, groupings at cluster-level would require hundreds of manual steps to replicate, and much more to construct a hyperscale datacenter.

E. *Simulation of Serverless Workloads*

Serverless computing encompasses cloud services that abstract operational concerns, such as resource provisioning and load-balancing, away from the user. They provide an event-driven interface, and charge users at a much finer granularity than the traditional cloud computing services.

Motivated by the promise held by serverless computing, the SPEC Research Group has investigated the properties of tens of real-world serverless platforms, and proposed the SPEC RG Reference Architecture for Function-as-a-Service (FaaS) [14]. This reference architecture is one of the first to provide a systematic approach to designing serverless platforms for which serverless computing is charged as a cloud service. OpenDC takes this high-level descriptive model, and adds to it a detailed, fine-grained, operational model. We validate and use this new capability in Section IV-A.

Figure 5 depicts the OpenDC detailed design for FaaS operations. At the core of this design are new components for function computation, function routing, and usage monitoring, which we discuss in the remainder of this section, in turn.

The *Computation* component provides all the logic surrounding the *Function Instance* (C), our model of a FaaS

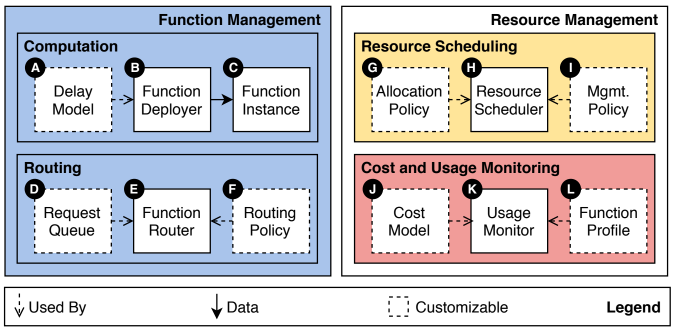

Figure 5: Architecture of the serverless model in OpenDC.

execution container. The *Function Deployer* (B) decides *how* the function instance is deployed on the available resources, on invocation. A special component, the *Delay Model* (A), simulates delays that typically occur during deployment, such as cold starts or lookup delays; we envision this will evolve as the community develops more advanced serverless platforms.

The *Routing* component is responsible for routing invocation-requests to available instances of their respective function. The *Function Router* (E) is the brain of this component. It uses a customizable *Request Queue* (D) to enqueue invocation requests, and a configurable *Routing Policy* (F) to select an available function instance to route a request to. OpenDC provides many classic policies, such as selecting a random available instance or the instance with the least cumulative idle time, which the community can complement.

The *Resource Scheduling* (H) component manages the datacenter resources on which the function-instances run. Users can configure it through two scheduling policies: the *Resource Management Policy* (I), which governs the lifetimes of function instances, and the *Allocation Policy* (G), which decides on the appropriate VM for each containerized instance.

To monitor individual functions, the *Usage Monitor* (K) operates on a map of *Function Profiles* (L). Each profile contains a selection of metrics, data structures, and other characteristic elements. The usage monitor employs a *Cost Model* (J) to determine the cost of computations using a variety of customizable cost functions.

F. *Simulation of TensorFlow*

Machine learning (ML) and deep learning have gained much recent attention due to their great potential in numerous areas [26], including speech recognition, medical image analysis, and product recommendation. Current ML applications can have large data and computational requirements, which makes cloud datacenters natural environments to execute them.

Among the many approaches developed to enable ML-use in complex applications, TensorFlow [15] is one of the most prominent and representative ML frameworks. Yet, this raises new challenges, such as data management. To explore and improve the operation of TensorFlow in datacenters, we extend OpenDC with a model for the TensorFlow ecosystem. Our detailed, fine-grained model captures TensorFlow workload

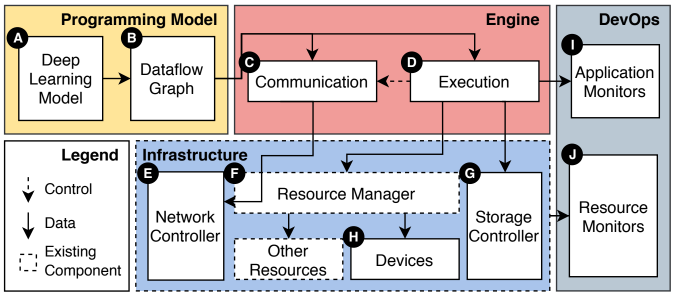

Figure 6: Architecture of the TensorFlow model in OpenDC.

execution and communication. We validate the model elsewhere[2]; Section IV-B showcases experiments using this model.

Figure 6 depicts the architecture of the TensorFlow model in OpenDC. The *Resource Manger* (F) allocates and deallocates resources using various policies. We use simple models for networking and storage: The *Network Controller* (E) is a simple network model for datacenters to control the dataflow between machines, considering bandwidth but not more complex network features. The *Storage Controller* (G) models persistent storage used during execution. We also provide an extension point, *Devices* (H), for heterogeneous resources.

Our TensorFlow model considers application, execution, and communication aspects. In OpenDC, to ensure generality beyond TensorFlow, an ML application can be modeled as a high-level *Deep Learning Model* (A) or a detailed *Dataflow Graph* (B). The *Execution* (D) components uses different strategies to orchestrate jobs across machines for distributed training (such as the parameter server strategy), and executes two types of operations (for mathematical computation and communication). The *Communication* (C) and Execution components collaborate to support different communication methods (e.g., asynchronous communication). The *Application Monitors* (I) record application-level metrics for TensorFlow users. Similarly, *Resource Monitors* (J) keep cluster-level metrics for sysadmins.

G. Software Engineering Process of OpenDC

We employ industry-standard development practices to develop OpenDC (R8). The main codebase is written in Kotlin, a modern and fast-growing programming language that is already adopted by large companies including Google [31]. Kotlin is designed to be fully interoperable with Java, and consequently it benefits from integration with the vast Java ecosystem. Kotlin greatly facilitates *co-routines*, allowing the simulator to efficiently and conveniently model datacenter operations—they simplify event-driven asynchronous execution thus fitting the simulation model, enable interruptions without thread-like context switching thus lowering the overhead, and enable concurrency even on a single OS-level thread (so, not parallelism) thus keeping complexity in check.

We enforce through *continuous integration* (CI) adherence to high development standards, running for each change an automated test suite and static code analysis tools (e.g., linting) to spot common mistakes. In addition, we employ a manual-review policy in our version control system, which requires an independent code review before changes to the simulator or its extensions can be integrated into the main codebase.

Similarly to the Linux project, we encourage and oversee the integration of OpenDC extensions into the main codebase. Although this approach increases the overall burden of maintenance, it also ensures high quality and compatibility across all components, as OpenDC evolves.

[2]MSc thesis Wenchen Lai, atlarge-research.com/pdfs/lai2020thesis.pdf

Table I: Use-cases that we cover in this work.

Sec.	Focus of use-case	Workload	Targets
IV-A	Serverless	[32]	Cold start, Cost
IV-B	Machine Learning	[33], [34]	Runtime, Energy Usage
IV-C	Resource procurement	[20]	CPU Contention
IV-D	Validation	[35], [36]	Workflow Makespan
IV-E	Education	Diverse	Students

IV. Use Cases for Datacenter Simulation

We present in this section five use-cases in which we cover relevant and emerging topics, and show how OpenDC is useful in such scenarios. Table I summarizes the use-cases.

A. Analyzing Serverless Workloads in Simulation

Despite efforts from the community to provide open-source implementations and architectures of FaaS (such as Open-Lambda [37]), exploring, testing, and evaluating different configurations of FaaS platforms remains challenging. For instance, the existing FaaS implementations are all specific to an architectural level, and are usually not trivial to configure for researchers that are not actively involved in the serverless field. Moreover, regardless of the technology used, cloud research often suffers from performance variability (e.g., due to heterogeneity of infrastructure) and from high cost to run experiments on physical infrastructure (as we argue in Section II-C). Lowering the threshold of entry to serverless research, OpenDC supports FaaS simulation (see Section III-E).

Experiment Setup: We focus in this experiment on reproducing the operation of the real-world FaaS platform provided by Microsoft Azure, as presented in their recent Hybrid Histogram experiments [32] on characterizing and optimizing real FaaS workloads. Beyond reproducibility, we aim to show OpenDC can help explore new scenarios with ease, so we add an experiment inspired by a recent change in how cloud operators charge FaaS users.

We model the environment to closely match the original execution environment, which includes 18 VMs of 2 cores and 4 GB of memory each. The container allocation and request routing policies are set to the default random policy, since they are not specified in [32]. We consider the two classes of resource management polices described in [32]: the Hybrid Histogram policy and the Fixed-Keep-Alive policies with 10 minute keep-alive intervals. We sample 68 applications from the first week of the Azure Functions trace, each with 1,500 to 2,500 function-invocations. During execution, OpenDC records the percentage of cold starts, per application; we have validated the results elsewhere, through a student thesis.

Table II: Cost comparison of running the Azure trace with the old 100 ms and the new 1 ms billing granularity.

Billing granularity	Every 100 ms	Every 1 ms	Difference
Price [$]	1.0168	1.0006	0.0162

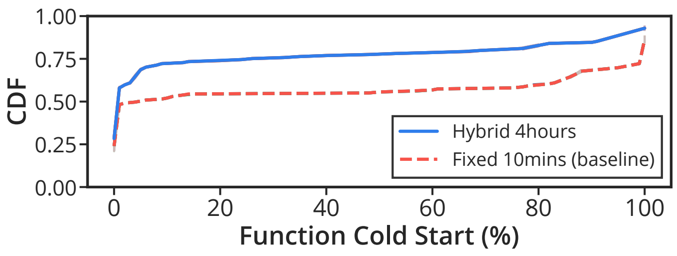

Figure 7: Cold start distribution of fixed keep-alive and hybrid policies in OpenDC Serverless. Fewer cold-starts are better.

Results: Figure 7 depicts the results of the simulations using both resource management policies. The cold start percentage trends closely follow those of [32, Fig.20]. The 4-hour Hybrid variant Hybrid Histogram policy allows for a consistent 10 to 20% reduction in cold starts at the 75th+ percentile. This supports [32]'s claim that the Hybrid Histogram policy is particularly effective for applications with infrequent invocation patterns, which usually reside in the last quarter.

Short-notice experiment on billing granularity: In December 2020—merely two weeks before the deadline of CCGRID—AWS Lambda changed the granularity of their function-billing, from 100 ms down to 1ms; other operators should follow soon. What would be the impact of this change on real users? Using OpenDC's customizable cost model interface (described in Section III-E), we ran a simulation on a sample of the first 7 days of the Azure trace (≈100,000 invocations). Table II compares the results for the 100 ms and 1 ms granularities. The cost difference between both models for this specific workload is ≈1.6%. The difference factor is low in this case, because most functions in the trace have execution times in multiples of a 100. The savings factor could be higher depending on the workload: data-streaming functions which typically execute for short duration could benefit from this change.

Conclusion: OpenDC is the first to capture the reference architecture for FaaS proposed by SPEC [14] in a simulation model. The results in this section show evidence the OpenDC serverless model is accurate enough to enable reproducing real-world results. We also conclude OpenDC can conveniently help compare the costs of running real workloads under different pricing models.

B. *Simulation for TensorFlow-based Machine Learning*

We experiment in this section with TensorFlow ML operations (see also Section III-F). We focus on understanding its performance in HPC environments, and in particular: (i) the influence of the distribution strategy on the dynamic operation of the system, (ii) the scalability of HPC environments in the presence of ML workloads, and (iii) the resource utilization in the system over time.

Table III: Total energy consumption (kWh). Workload: [34].

Strategy	Batch size	Number of Workers			
		1	2	4	8
Default	-	4.24			
Parameter Server	16	240.9	241.0	241.1	241.5
	32	120.5	120.5	120.7	119.4
	64	60.2	60.3	59.7	58.3
	128	150.6	148.5	144.5	136.4
Mirrored	16	240.9	241.0	241.1	241.5
	32	120.5	120.5	120.7	119.4
	64	60.2	60.3	59.7	58.3
	128	150.6	148.5	144.5	136.4

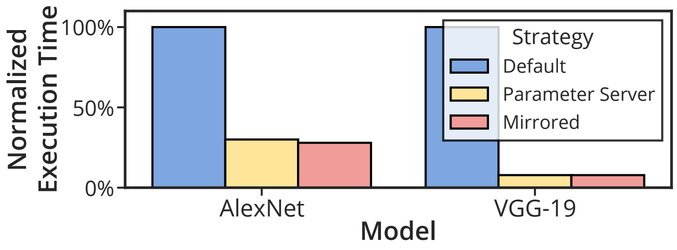

Figure 8: Strategy comparison on the Fanthom workload [33].

Experiment Setup: We reproduce experiments from the peer-reviewed papers [33] and [34]. For the A1 workload [34], we consider a single TensorFlow mini-application, which computes the AlexNet model and uses the Caltech 101 dataset; we execute this for its normal 142 iterations, varying the batch size between 16, 32, 64 and 128. For the Fathom workloads [38], we consider only AlexNet and VGG-19 using the ImageNet dataset, which we run for 200 iterations each. The arrival pattern of these two applications has no relation with the results, so we test them one by one. We recreate the HPC environment at KTH and the IBM "Minsky" platform, configuring the computing devices and network matrix from [34, Table 1] and [33, Figure 1], respectively. In our experiments, we test three distribution strategies: (i) *Default* executes computations on a single device, (ii) *Parameter Server* performs computation on multiple stateless worker devices, while a parameter server keeps track of and communicates the model parameters, and (iii) *Mirrored* enables synchronous distributed training on multiple GPUs, using a ring-AllReduce algorithm [15] to transfer model updates across devices. We report normalized execution time [33] and energy consumption, modeled linearly based on machine load [39], with an idle baseline of 90 W and a maximum power draw of 334 W.

Results: Figure 8 depicts the strategy comparison on the Fathom workloads in the IBM "Minsky" environment. For the AlexNet application, the normalized execution time (NET) of the application using the mirrored strategy is lower than that using parameter strategy. The VGG-19 application performs similar to parameter server strategy and mirrored strategy. A possible reason for this is that the VGG-19 application is computationally intensive, so its computation time accounts for a higher percent of the execution time.

Table III lists the simulation results of running A1 at KTH; we show here only the total energy consumption. We observe

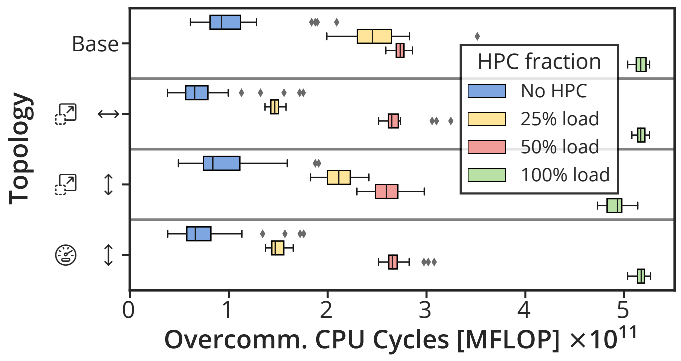

Figure 9: Overcommitted CPU cycles for a portfolio of *sampled HPC* workloads on a series of candidate topologies.
Legend: *horizontal* (↔) = more machines with fewer cores each, *vertical* (↕) = fewer machines with more cores each, *volume* (⤢) = more machines/cores, and *velocity* (◷) = higher clock speed of the cores.

that consumption varies significantly across setups. Compared to the default strategy with one worker, other settings have higher power-consumption. Moreover, the application consumes the least for a batch size of 64, compared to other batch sizes. We attribute this to AlexNet being optimized for its default batch size of 64.

Conclusion: OpenDC enables us to explore how different distribution strategies influence the performance and energy-consumption of TensorFlow applications.

C. Resource procurement for HPC-as-a-Service with Capelin

Long-term capacity planning (*procurement*) of cloud infrastructure is a critical yet non-trivial optimization problem that could lead to significant service improvements, cost savings, and environmental sustainability [40]. Although many approaches to this problem exist [41], [42], companies still rely on rule-of-thumb reasoning for decisions. To minimize operational risks, many such industry approaches lead to significant overprovisioning [43], or miscalculate the balance between under- and over-provisioning [44].

Experiment Setup: We explore in this experiment the question "what if a large part of the workload would be based on HPC jobs submitted by small and medium enterprises?"—a typical question for a national-level, public cloud provider. HPC jobs with this source are generally not very large, but can be CPU-sensitive; thus, we want to investigate CPU contention between VMs. We experiment with a one-month long trace of a business-critical workload from Solvinity, a Dutch cloud operator, of which an anonymized version has been published [20]. The trace consists of a subset of HPC VMs and other, conventional VMs. As the *baseline* scenario, we model the execution environment that ran the original workload. We then up-sample and down-sample the load incurred by the HPC part of the workload, from 0% (*No HPC*) to 100%. We conduct experiments with different *datacenter topologies*, the baseline and also new topologies that the cloud operator could procure.

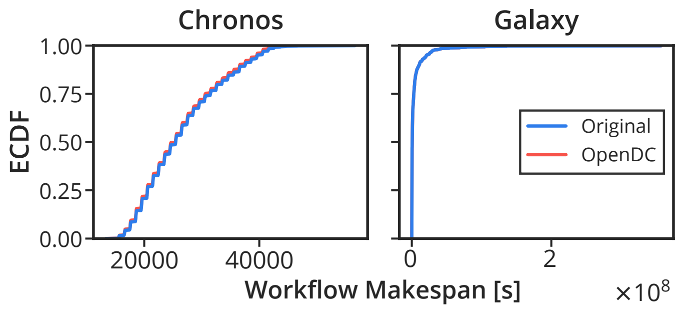

Figure 10: Empirical cumulative distribution function of workflow makespan for real-world and simulated execution of the Chronos [36] (left) and Galaxy [35] (right) workloads.

Results: We observe various metrics (OpenDC provides over 15), including the amount of overcommitted CPU cycles, which is a proxy for performance (higher is worse). Figure 9 depicts this metric. Changing the mix of HPC and non-HPC components in the workload has significant impact on the performance of the system, even when the total load is kept constant. We observe an increase of overcommitted CPU cycles relatively to the baseline by a factor of 5.58x in the worst case. Amongst the candidate topologies, the vertical volume scaling (⤢ ↕) performs best in scenarios with more HPC jobs, whereas horizontal scaling (↔) performs best in scenarios without HPC jobs.

Conclusion: These results highlight the impact of capacity planning on the performance of a system and show how OpenDC can be applied in such capacity planning scenarios.

D. Reproducibility with and validation of OpenDC

Reproducibility is a major challenge in datacenter experiments [13], [25]. We show how OpenDC can help.

Experiment Setup: We investigate in this experiment the operation of the workflow engine integrated in OpenDC, focusing on its ability to reproduce the execution of real-world workloads. We use two workloads: (i) Chronos [36], an industrial workload of workflows from a private cloud that processes monitoring data of industrial equipment, and (ii) Galaxy, a scientific workload of workflows executed for nearly two months on Galaxy Project's public Europe server [35], published in December 2020 by the Workflow Trace Archive [29]. We model the environment to closely match the original execution environments, using the peer-reviewed publications as main source of information [35], [36].

Results: Figure 10 depicts the empirical cumulative density functions (ECDFs) of the workflow makespan for the real-world and simulated execution of the Chronos (left) and Galaxy (right) workloads.

We focus first on the Chronos workload and observe that, although the simulated execution closely follows the structure of the real-world workload, there is a slight offset upward in the simulated makespan, resulting in an average relative error of 1.012%. We believe this issue is caused by small timing differences between the original execution environment of Chronos and OpenDC, for instance, because OpenDC models machine start-up and termination as instantaneous events.

For the Galaxy trace, the visual inspection of the results in Figure 10 (right) suggests that OpenDC's integrated workflow engine matches real-world execution. This is further confirmed by a two-sample Kolmogorov-Smirnov statistical test.

Conclusion: It remains a challenge to reproduce existing workloads without intricate knowledge of the execution environment. Still, we show that with OpenDC, we are able reproduce real-world workloads with high accuracy. We remain committed to the validation of the components of OpenDC and will continue to conduct reproducibility experiments.

E. Educating Diverse Students on Computer Systems

We envision the use of OpenDC for educational purposes, from informing (and enthusing) high-school students about computer systems, through educating junior researchers from the B.Sc.-level up, to giving M.Sc.-level students an instrument for scientific inquiry and engineering practice. We also want to engage in large-scale computer systems the groups of people that are currently under-represented. OpenDC is helping with various educational settings (R7):

- Already used to deliver key learning experiences in classroom-based (traditional) courses, as part of the Netherlands-wide doctoral course on Cloud and Big Data, Vrije Universiteit M.Sc. course on Distributed Systems, and TU Delft B.Sc. Honours Programme courses.
- Already used in project-based learning as part of credited activities in the M.Sc. and B.Sc. Honours Programmes at TU Delft and Vrije Universiteit Amsterdam, to engage top-level young students in scientific topics. Specifically, the practice of developing and using OpenDC is the focus of project-based learning on datacenters.
- Already used in project-based learning that is part of various M.Sc.-level credited activities at TU Delft and Vrije Universiteit Amsterdam, to engage several M.Sc. students that develop new datacenter techniques and conduct thorough datacenter research. This activity led to the TU Delft Best Graduate Award 2020[3].
- Already integrated in workshops held as part of the Restart.network[4] education network for refugees, unemployed single mothers, and others in the Netherlands.
- Demonstrated in national conferences with industry and academia, such as ICT.OPEN and CompSys NL.

V. Related Work

In this section, we survey related work, for which we summarize the comparison in Table IV. The community has already built many high-quality simulators that provide a rich set of features to build upon [54], [55]. We select here the simulators closest in nature to OpenDC. The others offer typically a single feature, or are very general and thus require repeating the work we propose here for each specific model.

Overall, OpenDC proposes unique modeling advances, such as (i) the serverless model that is the first to detail the reference architecture proposed by SPEC [14] into an operational model, (ii) a detailed model for machine learning workloads based on TensorFlow, and (iii) prefabricated components for sharing designs. The integrated nature and the convenience features of OpenDC allow it to be deployed in practice quickly, even without deep expertise.

Table IV: Comparison of selected datacenter simulators.

Project	Environment	Stakeholders	Highlighted Feat.	GUI
CloudSim [7]	Cloud, Edge, Fog [45]	Research	VC*, N, S, E, WF† [46], FD†, EXP†, CM, PI†	✓† [47]
SimGrid [8]	Grid, Cloud [48], P2P	Research, Edu. [49]	VC*† [50], N*, S, E*, WF* [51]	✓† [51]
DGSim [52]	Grid	Research	WF, F, EXP	✗
GroudSim [53]	Grid, Cloud	Research	WF, CM, F	✗
iCanCloud [9]	Cloud	Research	VC, N*, S, CM	✓*
OpenDC (this work)	Cloud	Research, Education	VC*, N, S, E, CM, FS*, ML, WF, F*, PI, EXP*	✓*

Models: VC = VMs and containers, N = Network, S = Storage, E = Energy, CM = Cost models, FS = FaaS, ML = Machine learning, WF = Workflows, FD = Federation; **Phenomena**: F = Failures, PI = Performance interference, **Tools**: EXP = Experiment automation. **Support**: ✗= No, ✓= Yes.
$\dagger$ = extension, not integrated; $\star$ = advanced, carefully calibrated feature.

Closest alternative for research and development processes: OpenDC is closest in nature to the CloudSim ecosystem. CloudSim includes a high-quality simulator [7], which focuses on simulating cloud system components including virtual machines, data centers, and resource provisioning policies. It also includes numerous other single-feature simulators, such as iFogSim [45], WorkflowSim [46], and CloudAnalyst [47]. However, the single-feature simulators extend CloudSim each in their direction and cannot be combined without extensive engineering. In contrast, OpenDC offers an integrated approach, and specific modeling advances for its main features.

Similar is the SimGrid framework [8], which serves as the foundation of many simulators, such as SimGrid VM [50], Schlouder [48] and WRENCH [51]. In contrast to OpenDC, it is more general purpose (supporting not only clouds, but also P2P networks) and runs at a much finer granularity, which in turn enables emulation of specialized applications (e.g., MPI). However, SimGrid and its ecosystem do not provide the advances described in the overall part of this section.

Alternatives for training and education: The use of simulation in education is generally not documented, but we assume it to be widespread. The notable emergence of WRENCH [49] shows the potential of simulation for teaching complex subjects in the computer science curriculum. OpenDC provided already extensive engagement with many categories of students and lengths and complexities of projects (see Section IV-E), which have not been reported elsewhere.

VI. Conclusion and Future Work

Cloud infrastructure is in the backbone of our digital society. To satisfy the growing demand, datacenter architects must

[3]TU Delft Best Graduate Award 2020. bit.ly/AndreadisTUDAward
[4]https://restart.network/

address complex challenges in designing, operating, and using modern datacenters; simulation is key. Although the community has already developed many powerful simulators, emerging new technologies and applications in clouds, and the need to educate more people, raise new challenges. We propose the OpenDC 2.0 datacenter simulation platform, and present five representative use-cases to demonstrate its benefits.

In this work, we have designed and implemented OpenDC. Novel, our model integrates support for serverless computing and TensorFlow-based machine learning. OpenDC also provides a convenient web interface for interactive experimentation, support for experiment automation, a library of prefabs for constructing and sharing datacenter designs, and support for diverse metrics and input formats. We demonstrate the benefits of OpenDC through five representative use-cases: serverless, machine learning, procurement of HPC-as-a-Service infrastructure, reproducibility, and educational practices. We highlight the findings related to serverless computing and HPC-as-a-Service procurement as novel.

Overall, we showed that OpenDC helps understand how datacenter works, design datacenter infrastructure, and train the next generation of datacenter engineers. OpenDC is now released as FOSS, at: https://github.com/atlarge-research/opendc

Acknowledgments: NWO MagnaData and OffSense.

References

[1] Gartner Inc., “Gartner forecasts worldwide public cloud revenue to grow 17% in 2020,” Press Release, 2019.

[2] Tannenbaum *et al.*, *Condor: A Distributed Job Scheduler*. MIT Press, 2001.

[3] Boutin *et al.*, “Apollo: Scalable and coordinated scheduling for cloud-scale computing,” in *OSDI*, 2014.

[4] Thain *et al.*, “Distributed computing in practice: the Condor experience,” *CCPE*, vol. 17, 2005.

[5] Ghanbari *et al.*, “Optimal autoscaling in a IaaS cloud,” in *ICAC*, 2012.

[6] Klusácek and Tóth, “On interactions among scheduling policies: Finding efficient queue setup using high-resolution simulations,” in *Euro-Par*, vol. 8632, 2014.

[7] Calheiros *et al.*, “CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms,” *SPE*, vol. 41, 2011.

[8] Casanova *et al.*, “Versatile, scalable, and accurate simulation of distributed applications and platforms,” *JPDC*, vol. 74, 2014.

[9] Nuñez *et al.*, “iCanCloud: A flexible and scalable cloud infrastructure simulator,” *JGC*, vol. 10, 2012.

[10] K. S. Trivedi, *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*. Prentice Hall, 2016.

[11] Harchol-Balter, *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*. Cambridge University Press, 2013.

[12] R. Jain, *The Art of Computer Systems Performance Analysis*. Wiley, 1991.

[13] Ivie and Thain, “Reproducibility in scientific computing,” *CSUR*, 2018.

[14] Eyk *et al.*, “The SPEC-RG reference architecture for FaaS: From microservices and containers to serverless platforms,” *Internet*, vol. 23, 2019.

[15] Abadi *et al.*, “TensorFlow: A system for large-scale machine learning,” in *OSDI*, 2016.

[16] Iosup *et al.*, “The OpenDC vision: Towards collaborative datacenter simulation and exploration for everybody,” in *ISPDC*, 2017.

[17] J. Banks *et al.*, *Discrete-Event System Simulation*. Pearson, 2014.

[18] Andreadis *et al.*, “A reference architecture for datacenter scheduling: design, validation, and experiments,” in *SC*, 2018.

[19] Baset *et al.*, “Towards an understanding of oversubscription in cloud,” in *Hot-ICE*, 2012.

[20] Shen *et al.*, “Statistical characterization of business-critical workloads hosted in cloud datacenters,” in *CCGrid*, 2015.

[21] Iosup *et al.*, “Massivizing computer systems: A vision to understand, design, and engineer computer ecosystems through and beyond modern distributed systems,” in *ICDCS*, 2018.

[22] Beek *et al.*, “A CPU contention predictor for business-critical workloads in cloud datacenters,” in *SASO*, 2019.

[23] Koh *et al.*, “An analysis of performance interference effects in virtual environments,” in *ISPASS*, 2007.

[24] Gallet *et al.*, “A model for space-correlated failures in large-scale distributed systems,” in *Euro-Par*, vol. 6271, 2010.

[25] Uta *et al.*, “Is big data performance reproducible in modern cloud networks?” in *NSDI*, 2020.

[26] C. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[27] Korte et al., “e-Skills for jobs in Europe: Measuring progress and moving ahead,” European Commission report., Sep 2014.

[28] Iosup *et al.*, “The Grid Workloads Archive,” *FGCS*, vol. 24, 2008.

[29] Versluis *et al.*, “The Workflow Trace Archive: Open-access data from public and private computing infrastructures,” *TPDS*, vol. 31, 2020.

[30] Feitelson *et al.*, “Experience with using the Parallel Workloads Archive,” *JPDC*, vol. 74, 2014.

[31] JetBrains s.r.o., “The Kotlin programming language,” Press Kit., 2020.

[32] Shahrad *et al.*, “Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider,” in *ATC*, 2020.

[33] Guignard *et al.*, “Performance characterization of state-of-the-art deep learning workloads on an IBM ”Minsky” platform,” in *HICSS*, 2018.

[34] Chien *et al.*, “Characterizing deep-learning I/O workloads in TensorFlow,” in *PDSW-DISCS*, 2018.

[35] Afgan *et al.*, “The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2018 update,” *NAR*, vol. 46, 2018.

[36] S. Ma *et al.*, “Ananke: A Q-Learning-based portfolio scheduler for complex industrial workflows,” in *ICAC*, 2017.

[37] Hendrickson *et al.*, “Serverless computation with OpenLambda,” in *HotCloud*, 2016.

[38] Adolf *et al.*, “Fathom: reference workloads for modern deep learning methods,” in *IISWC*, 2016.

[39] Blackburn and Grid, “Five ways to reduce data center server power consumption,” *The Green Grid*, vol. 42, 2008.

[40] Barroso *et al.*, *The Datacenter as a Computer*. Morgan & Claypool Publishers, 2018.

[41] Zhang *et al.*, “R-Capriccio: A capacity planning and anomaly detection tool for enterprise services with live workloads,” in *Middleware*, 2007.

[42] Carvalho *et al.*, “Capacity planning for IaaS cloud providers offering multiple service classes,” *FGCS*, vol. 77, 2017.

[43] Glanz, “Data Centers Waste Vast Amounts of Energy, Belying Industry Image,” *N.Y. Times*, 2012.

[44] Nazareth and Choi, “Capacity management for cloud computing: A system dynamics approach,” in *AMCIS*, 2017.

[45] Gupta *et al.*, “iFogSim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments,” *SPE*, vol. 47, 2017.

[46] Chen and Deelman, “WorkflowSim: A toolkit for simulating scientific workflows in distributed environments,” in *e-Science*, 2012.

[47] Wickremasinghe *et al.*, “CloudAnalyst: A CloudSim-based visual modeller for analysing cloud computing environments and applications,” in *AINA*, 2010.

[48] Michon *et al.*, “Schlouder: A broker for IaaS clouds,” *FGCS*, vol. 69, 2017.

[49] Tanaka *et al.*, “Teaching parallel and distributed computing concepts in simulation with WRENCH,” in *EduHPC*, 2019.

[50] Hirofuchi *et al.*, “SimGrid VM: Virtual machine support for a simulation framework of distributed systems,” *TCC*, vol. 6, 2018.

[51] Casanova *et al.*, “Developing accurate and scalable simulators of production workflow management systems with WRENCH,” *FGCS*, vol. 112, 2020.

[52] Iosup *et al.*, “DGSim: Comparing grid resource management architectures through trace-based simulation,” in *Euro-Par*, vol. 5168, 2008.

[53] Ostermann *et al.*, “GroudSim: An event-based simulation framework for computational grids and clouds,” in *Euro-Par*, vol. 6586, 2010.

[54] Bambrik, “A survey on cloud computing simulation and modeling,” *SNCS*, vol. 1, 2020.

[55] Byrne *et al.*, “A review of cloud computing simulation platforms and related environments,” in *CLOSER*, 2017.