Vrije Universiteit Amsterdam

Bachelor Thesis

Sunfish: Enabling Predictive Analytics for Datacenters Through Digital Twinning

---

Author: Mateusz Kwiatkowski (2805533)

1st supervisor:	Prof. Dr. Ir. Alexandru Iosup
daily supervisor:	drs. Dante Niewenhuis
2nd reader:	drs. Matthijs Jansen

A thesis submitted in fulfillment of the requirements
for the VU Bachelor of Science degree in Computer Science.

July 22, 2026

Abstract

In the modern AI economy, the strong computational demand causes datacenters to become complex, diverse facilities. The sheer volume of CPUs, GPUs, NPUs *etc.* necessary to satisfy the customers' needs complicates system administration. Moreover, future warehouses are at an even higher risk of becoming unmanageable, according to the Jevon's Paradox of Computer Systems. To address this problem, the scientific community has proposed digital twinning as a datacenter management tool. Digital Twins, which mirror complex objects and processes to provide actionable management improvements, are a novel way to tackle the rising warehouse complexity. However, Datacenter Digital Twins are still under development, and lack crucial features, such as predictive analytics. Without predictive maintenance and forecasts, system administrators cannot make well-informed operational decisions.

In this work, we propose to enable predictive analytics for datacenters using digital twinning. We survey the datacenter digital twinning field, and organize our findings into a system model. Based on real-world use-cases we design *Sunfish* – a novel reference architecture for predictive datacenter digital twins, and evaluate it through prototype-based experiments. Our results indicate *Sunfish* is capable of reliably differentiating between mild and severe compute failures, and can successfully incorporate a predictive analytics engine to the benefit of datacenter managers. The main findings are backed-up by experiments which replicate a peer-reviewed publication evaluation approach to show that *Sunfish* can notify datacenter operators of compute failures. Moreover, to show that the system can incorporate a predictive analytics-engine, we have conducted a novel, conceptual experiment to exhibit the potential future gains of a holistic Datacenter Digital Twin.

Acknowledgements

I wish to express my thanks to the many people who have helped me in preparation of this thesis.

Dante Niewenhuis, my daily supervisor, with patience and understanding mentored me during our collaboration, which led to the timely and successful completion of this thesis.

Alexandru Iosup, my 1st supervisor, provided invaluable feedback and suggestions that helped immensely during the design and writing.

Jesse Donkervliet guided the thesis timeline and greatly assisted in preparation of the final defence during the weekly thesis meetings.

Daniel Halasz provided many insightful comments which shaped the technical contributions.

Daniele Bonetta, Matthijs Jansen, Tiziano De Matteis, Radu Nicolae, Jure Antunović, Sacheendra Talluri, and Krijn Doekemeijer of the AtLarge Research Group provided indispensable knowledge on conducting good Computer Systems research over the last 2 years.

Most importantly, I would like to thank Waldemar, Anna and Julia for their love and support while I was working on this thesis.

Dedication

To my sister Julia

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Currently, computer and network systems play a crucial part in the digital industry. For example, the transport, education and government sectors largely depend on digital services, which are hosted in datacenters [1]. Datacenters are complex facilities, housing thousands of servers. Moreover, the advancements in Artificial Intelligence have sped up data datacenter expansion. To address the recent rise in demand for computation, datacenter managers add new components and more heterogeneous architectures (*e.g.,* GPUs, NPUs) [2] to the already complex warehouses. However, in return datacenter complexity increases significantly. To make better operational decisions despite the massive scale, promising technologies arise such as Datacenter Digital Twins [3].

Datacenters house large volume of computers for processing and storage of data from various organizations and fields of activity. Over 3 million jobs in the Netherlands directly depend on cloud services, which are hosted in datacenters. Since many public services continue to move online (*e.g.,* online administration and taxation, education), the fraction of Dutch professionals who depend on the cloud for work will exceed 35% by 2025 [1].

In the modern Artificial Intelligence (AI) economy, datacenters need diverse and scalable server architectures, because inference-based workloads require more heterogeneous server components (GPUs, TPUs, NPUs *etc.*) to perform well. In return, operating a modern datacenter warehouse with thousands of diversified servers presents a difficult challenge that requires fast and well-informed decisions from on-site engineers.

The computational requirements of AI are expected to increase in the future [3]. Because of this, datacenter complexity will continue to grow, and it will become more difficult to manage [4]. Future servers and software related services to them will include even more specialized hardware, which, while improving datacenter performance, will exhibit behaviour that is harder to predict. Already the rapid expansion of datacenters has increased the presence of service failures across all cloud services [5]. Preventing failure-caused outages in advance could help datacenter operators reduce operational costs, as over 20% of all reported outages amount to more than 1 million US$ [6].

In short, the high computational demand of AI and the end of Dennard's scaling have resulted in the rise of larger and more heterogeneous datacenter architectures [2]. Both

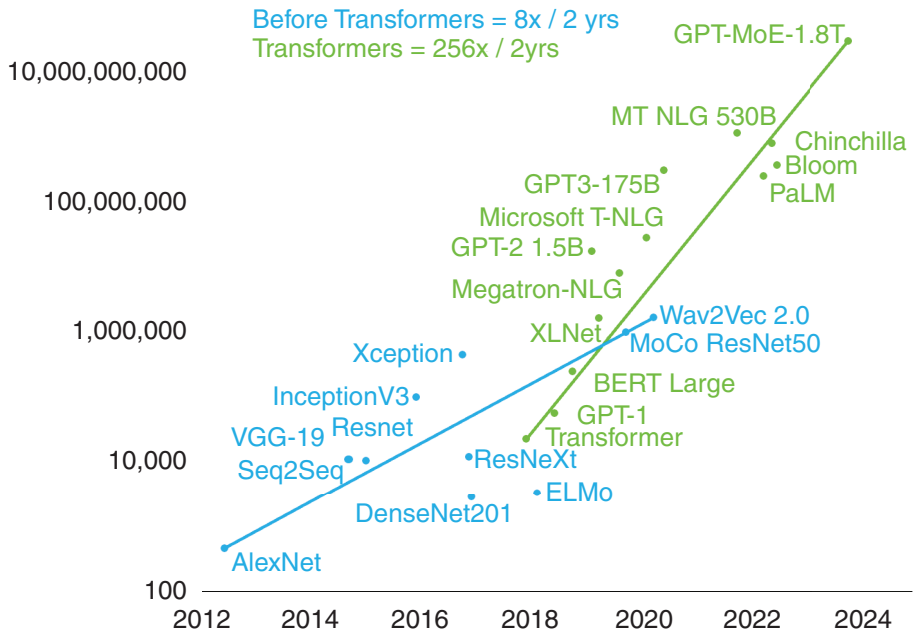


Figure 1.1: Explosive growth in AI computational requirements drives datacenter upgrades (source: NVIDIA Analysis: reproduction with NVIDIA permission by [3]). X-axis presents the year, and the Y-axis presents the training compute requirement (in petaFLOPs). The plot shows the one of the reasons behind why the complexity of datacenters is rising.

events create a need for more careful datacenter management to tackle the unprecedented complexity and ensure availability of all cloud services. To address this new problem a concept of a datacenter Digital Twin (DT) was proposed [3].

1.1 Context

A DT is a virtual model of an intended or actual real-world system that serves as its counterpart for purposes such as simulation, integration, testing, monitoring and maintenance. The digital twin replicates the physical system to predict failures, prescribe real-time actions for mitigating unexpected events, observing and evaluating the behaviour of the system [7].

Most modern DT usages are related to prognostics and system health management [8]. For example, in aerospace engineering, the DT analyzes operational data (*e.g.,* temperature, vibration) to predict when a airplane component is likely to fail. The DT can reliably manage the health of the physical entity by detecting fatigue cracks on aircraft wings or damage to the wind turbine blades [9]. This allows maintenance to be scheduled proactively, reducing unplanned downtime and preventing catastrophic failures. Forecasting future maintenance and managing the physical health of an object or facility are the prime purpose of many DTs used in practice [10].

The concept of a DT began in 1960s, at the National Aeronautics and Space Agency (NASA) [11]. NASA pioneered the concept in order to debug issues with its spacecraft. However, the term "digital-twin" dates back to 2003, when Dr. Michael Grieves of Dassault Systémes introduced the 3 core components of a DT: the virtual entity, physical entity and the two-way connection (see Figure 1.2). Due to insufficient technological foundations, little work is available on DTs between 2003 and 2018, and it is only with the rapid growth of cloud computing, Internet-of-Things (IoT) and Big Data analytics that DTs have re-emerged. Today, research is focused on bridging the gap between the long-established foundations of DTs and new, novel applications in academia and industry, such as the Datacenter Digital Twin (DCDT) [8, 3].

A DCDT mirrors the structure, context and behaviour of a datacenter [3]. The foundation to any digital twin is good monitoring and sensing capabilities in the physical entity. Datacenters, meet this requirement easily because they already connect hundreds of monitoring sensors. With hundreds of gigabytes of useful information coming from distributed IoT sensors inside the warehouse, we can gain insight into failure patterns, energy usage, heat dissipation *etc.* What remains challenging is to connect the physical and virtual spaces with a bi-directional connection and to use the monitoring insights and data analysis results for autonomous decision-making. Crucial to DCDT operation are predictive capabilities and the continuous interaction with the real-world datacenter.

There already exist DCDT deployments. For example, ExaDigiT [12] is a framework for digital twin development of supercomputers. It has been demonstrated at the Frontier supercomputer and it facilitates virtual prototyping and system optimization.

Nonetheless, existing DCDT's are still very limited in their capabilities as the definition and scope of a DCDT concept is shallow and unclear. After all, only recently did the hardware capabilities needed to continuously simulate a datacenter become available [8]. Many DCDT frameworks still lack critical data analysis components, fault detection mechanisms, profiling techniques *etc.* [13], rendering them unusable in large-scale systems. Such limitations gravely reduce the applicability of DCDT's in real world scenarios [1]. DCDT's are urgently needed, because datacenters exhibit hundreds unexpected events every day, such as *e.g.,* service failures or hardware faults. Downtime, which is the result of failures, disturbs the users and produces unfulfilled Service Level Agreements (SLA) [5]. However, predicting datacenter behaviour quickly and reliably is a non-trivial problem that remains insufficiently unaddressed in the existing DCDT architectures [13, 3] and deployments [12].

1.2 Problem statement

In this work, we address the lack of a unified DCDT system model and the absence of predictive capabilities in existing DCDT system designs. We argue that the current state-of-the-art DCDT's lack sufficient predictive capabilities that are essential to real-time facility management of a modern datacenter. Because the main purpose of many DTs is to forecast the short and long-term facility behaviour, DCDT without predictive capabilities cannot maintain the health of the datacenter effectively. We posit that including holis-

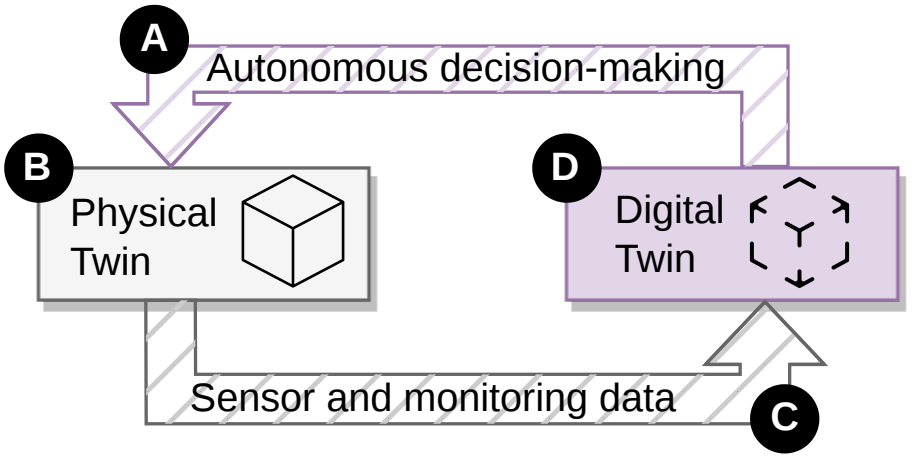


Figure 1.2: Elements of the digital twin ecosystem [14] include: the insights and decisions coming from the digital twin (Ⓐ), the physical infrastructure (Ⓑ), the data coming from the physical twin telemetry (Ⓒ), and the digital counterpart to the physical twin (Ⓓ). This thesis focuses on components (Ⓐ), (Ⓒ), and (Ⓓ) in this ecosystem, proposing design improvements to (Ⓓ, Ⓒ), and the feedback loop (Ⓐ).

tic predictive analysis in DCDT design can aid in efficient datacenter management and prevent missing SLA's. For example, preventing compute failures could greatly benefit datacenter operators. To enable insights from both historical data and immediate telemetry, we propose that digital twinning can be enhanced by integrating predictive analytics through Operational Data Analysis (ODA). We envision DCDT's as systems indispensable in future datacenters, actively interacting with the real-world facility, lowering operational costs and predicting hardware failure and software faults. Our solution to this problem encompasses different levels of ODA (*e.g.,* in-band analytics, out-of-band analytics) for holistic datacenter modelling. Together with a unified DCDT system model and a revolutionary evaluation method of DCDT's, we hope to bring the modern vision of DT's to datacenters.

1.3 Research Questions

Main Research Question: How to enable predictive analytics in datacenters through digital twinning?

We divide the problem of designing a predictive DCDT into three research questions:

RQ₁ *How to assess the current state-of-the-art of digital twinning for datacenters?*
There is currently a lack of a unified system model of what constitutes a DCDT, and the differences between existing DCDT deployments. To alleviate this problem, we aim to develop a holistic DCDT model that factors in the necessary components of a DT. This is a challenging, because the DCDT system model must address many kinds

of operational and technical requirements, compatible with the existing background on DTs.

RQ₂ *How to design a DCDT reference architecture using discrete-event simulation and predictive data analysis?*
Existing DCDT frameworks lack the necessary predictive capabilities to prevent unplanned behaviour in datacenters [13, 12, 15, 3]. In this work, we aim to explore the design space of a predictive DCDT and the different design trade-offs. Through discrete-event simulation, we aim provide the foundation for the system to interact with a physical datacenter. What makes the task challenging are the many functional and non-functional requirements of a DCDT that need careful consideration. The architecture must comply with the generic DT model and address the non-trivial challenges in operating a modern datacenter.

RQ₃ *How to evaluate and validate a DCDT reference architecture in relation to system requirements*?
To understand the operation of the proposed system and whether it meets its design goals we need to measure it's performance. This is a challenging and non-trivial task that requires a careful design of a set of experiments that faithfully show the system at work. Additionally, we need to address the novel challenge of overcoming the lack of a physical twin to experiment with.

1.4 Research Methodology

To answer *RQ₁* we conduct a literature review as proposed by *Kitchenham et al.* [16] along with the guidance of the supervisor. Firstly, we determine the right review method. Secondly, we identify the various works related to DCDT's using different search strings (*e.g.,* "Datacenter Digital Twinning", "ICT Virtual Twin") and query combinations ("Datacenter `AND` Maintenance"). To search for the results we use the digital libraries of Google Scholar, DBLP, ACM Digital Library, IEEExplore, Springer *etc.* Thirdly, we select work relevant to our research and organize the details of each article into a table. A potential outcome of this could be a system model for DCDT's. We envision the literature review supplying us with potential use-cases for the predictive DCDT. Based on the found use-cases, we brainstorm and formulate the functional and non-functional requirements for the predictive DCDT reference architecture.

To answer *RQ₂* we closely follow the *AtLarge Design Process* [17] under the guidance of the supervisor. A potential outcome might be a simulation-based DCDT system that meets the requirements listed as a part of *RQ₁*. Firstly, following the literature review, we list the functional and non-functional requirements of a predictive DCDT. We specify the pragmatic and innovative design possibilities to include in the reference architecture. The designed system builds upon the `OpenDC` platform for datacenter simulation [18], extending it with predictive analysis capabilities. Lastly, we ensure that the design is scientific and testable and can be evaluated with comprehensive experiments.

To answer RQ_3 we implement a prototype of the designed reference architecture. We design comprehensive experiments that evaluate and validate the prototype based on the reference architecture. We first gather a set of questions worth asking about the performance and impact of the predictive DCDT and then set out to answer them with the prototype. We define the correct experiment setup(s) and perform the experiments on a specified hardware, considering different usage scenarios.

1.5 Thesis Contributions

1. **Conceptual**:

   C_1 We conduct a comprehensive literature review and detailed analysis of existing works on digital twinning in the scientific research community. We collect and organize the DCDT's characteristics and based on our findings we propose a unified system model of the design space.

   C_2 We propose the design of *Sunfish*, a discrete-event DCDT for reliable and timely failure prediction in datacenters. *Sunfish* includes a set of novel system components which leverage ODA and discrete-event simulation.

   C_3 We evaluate *Sunfish* using a novel experimentation technique and datacenter workload traces from the industry. We design a method to evaluate DCDTs without expensive and costly real-world experimentation. We conduct a set of exhaustive experiments and analyse the results.

2. **Technical:**

   C_1 We prototype *Sunfish* following the established DT design principles using discrete-event simulation and ODA. We include the code as an Open Science artifact and ensure the prototype remains accessible to the broader scientific community including detailed project documentation.

   C_2 We provide the experiment setup, validation and evaluation of *Sunfish* for detecting and predicting datacenter failures in real-time as an Open Science artifact.

1.6 Academic Integrity Declaration

1.6.1 Non-Plagiarism Declaration

I hereby declare that this thesis is my own independent work and writing. The thesis does not contain any material copied from other sources (person, Internet, or AI), and has not been submitted for assessment elsewhere. I acknowledge that the usage of material from other works or paraphrase of such material without proper citations or credit will be treated as plagiarism. I declare that this thesis is free from AI generated content and has been written without the help of any AI tools. To order to adhere to the strictest

restrictions on AI-usage in higher education, this work follows the Berkley School of Law Artificial Intelligence Policy, as stated in https://www.law.berkeley.edu/wp-content/uploads/2026/05/AI-Final-Policy-26.pdf.

1.6.2 Preventing Reference Fraud

I hereby declare that all the references in this thesis refer to genuine scientific work published in peer-reviewed journals or other sources of reliable and safe online information (*e.g.,* Wikipedia articles) and have been used in accordance to the article authors' wishes. Additionally, under the guidance of the supervisor this work adheres to the strictest rules for referencing and to prove the originality of all references, each BibTeX citation contains a `note` field with the following comment: *This BibTeX citation comes from:* followed by the URL leading directly to the citation source. In case of citations not formatted in BibTeX, the same format follows but with adequate reference-style name (*e.g.,* APA, Chicago, MLA).

1.7 Societal Impact Including Open Science Artifacts

Any program that is difficult to understand and reason about is sure to accumulate technical debt. However, sometimes large-scale systems can be complex and hard to comprehend inherently. In such scenario, software that can aid system management is necessary. Computer Systems, more complex now than ever before, must remain accessible to be beneficial to the digital society. This work addresses the four grand societal challenges related to this goal: (1) manageability (2) responsibility (3) sustainability (4) usability [1]. *Sunfish* addresses (1) directly by making large-scale datacenter management easier. We address (2) by ensuring our work adheres to the Findability, Accessibility, Interoperability and Reusability (FAIR) principles of Open Science. Moreover, in this thesis we try to make DCDT systems more understandable to the broader scientific community by providing a unified system model. Additionally, we contribute to responsible software design by adhering to best software engineering practices in the design of the prototype. (3) is addressed indirectly, as the consequences of the insights provided by a holistic, ODA powered DCDT can help datacenter managers make decisions that are more sustainable in the future. We contribute to (4) by helping predict unexpected failures and lowering operational costs, ensuring datacenters can continue to be usable in the future. We believe this work has a strong societal impact due to addressing the four grand societal challenges described by Iosup *et al.* and we hope through this work we can advance the scientific research community towards a more sustainable future. Abiding the FAIR data principles, the entire source code of the prototype and related work has been made available at the https://github.com/M-J-Kwiatkowski/opendc and the https://github.com/M-J-Kwiatkowski/sunfish repositories. The reuse and reproduction of experiments is explained in a detailed guide at the root of the repository, along with the necessary dependencies and experimental setup.

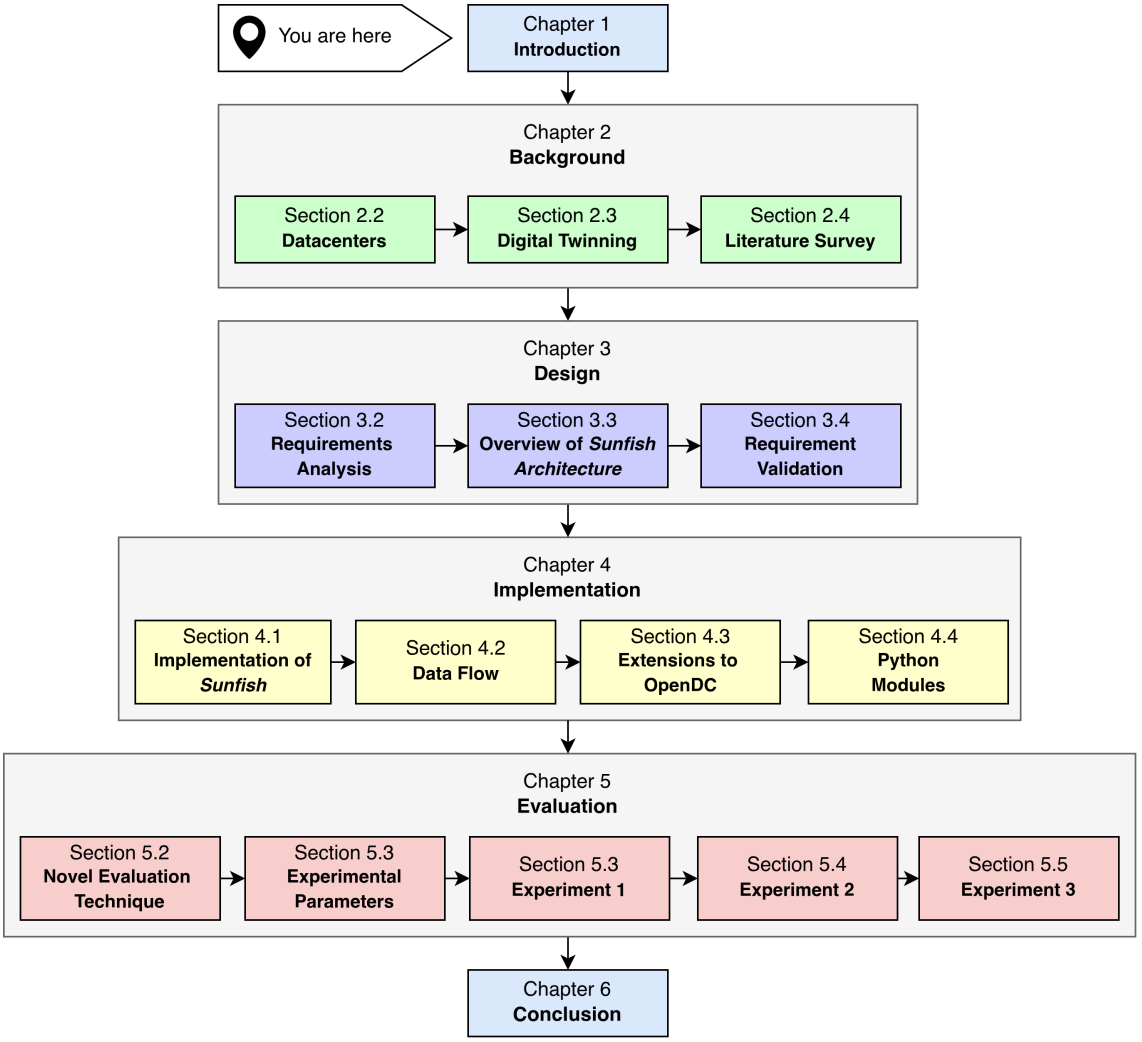


Figure 1.3: Structure of this thesis, with suggested reading flows.

1.8 Thesis Structure

The remainder of the thesis is structured as depicted in Figure 1.3. In Chapter 2, we describe the relevant background information. In Chapter 3, we present the design of *Sunfish*. In Chapter 4 we present the technical details of *Sunfish* prototype. In Chapter 5 we evaluate the prototype of the system and validate it against the set of functional and non-functional requirements. In Chapter 6 we conclude the thesis with a summary of contributions and potential future work.

Chapter 2

Background

2.1 Overview

The contribution in this chapter is three-fold:

- C_1 We provide a brief overview on datacenters (Section 2.2) datacenter simulation (Section 2.4), compute failures (Section 2.3), and digital twinning (Section 2.5).
- C_2 We survey the state-of-the-art concerning datacenter digital twinning (Section 2.6.2).
- C_3 We construct a system model for existing datacenter digital twins (Section 2.6.3)

2.2 Datacenters

In this section we provide a short background on datacenters, datacenter simulation and compute failures. We find it useful to provide a brief introduction to these topics so as to ensure reader's fullest understanding of subsequent chapters.

A datacenter is "a physical room, building, or facility for the purpose of the storage, management, and dissemination of data and information, including training artificial intelligence, housing IT infrastructure, computer systems, and associated components." [19]. In essence, datacenters contain a large amount servers, and everything that is needed to maintain them. Most often servers are specially-designed motherboards with a (multicore) Central Processing Unit (CPU), Random Access Memory (RAM) and storage. More diverse servers include a CPU, Tensor Processing Unit (TPU), or Neural Processing Unit (NPU). To efficiently organize the datacenter, servers are placed within server *racks*. To maintain a large number of server racks, datacenters contain a cooling system to control the heat transfer and temperature of both the hardware and the entire facility. Additionally, datacenters consume vast amounts of electricity [19]. Because of this, the datacenter

Figure 2.1: Example of a datacenter in CERN, Switzerland (2010) [19]. In the figure we can see servers within servers racks, and the network cables.

power supply play a critical role in keeping the services running on the servers always available. An example datacenter in Conseil Européen pour la Recherche Nucléaire (CERN), is depicted on Figure 2.1.

Datacenters form the backbone of the digital society. The main stakeholders, besides the companies in the Information Technology (IT) sector, are intelligent healthcare, remote work, online gaming, digital government and education, banking and finance, transport and logistics [1]. All of the above industries need reliable datacenters to work well in the 21st century.

The high demand for online services drives datacenter complexity. Moreover, due to the Jevon's paradox of Computer Systems [20], improved availability increases the demand. As a result, datacenters contain hundreds, or even thousands of hardware components. Every device may have a different vendor, new configuration, unusual interface *etc.* Because of this, datacenter operators are often faced with difficult operational and architectural challenges [21, 18], which span software and performance engineering. Making sure that all the parts of the datacenter work together is a tough task. What drives datacenter complexity even further is that sophisticated systems are not merely a sum of their parts [22]. The combination of the above factors makes datacenter management a difficult, non-trivial challenge.

2.3 Compute Failures

A failure is defined as "an event that makes a system fail to operate according to its specifications" [23]. A simple example of a failure is when an old hard drive stops working. Data on the disk is lost, and services running on the respective server are disrupted. In reality, problems with the power supply account for most failures (54%). The runner-ups are problems with cooling (13%), and IT/software (12%) [6]. Power related failures may stem from software/firmware issues, battery degradation, overheating, power generator failure, mechanical problems, faulty control logic *etc.* [6].

Failure-caused outages are costly. According to the Uptime Institute, 20% of all outages cost more than 1 million USD$ [6]. Moreover, failures in datacenters result in service downtime, missed Service Level Agreements (SLA) and user inconvenience [5, 23]. Industries that rely on 24 hour access suffer the most from datacenter outages. The impact of failures on medical informatics, nuclear power-plants, banks and financial institutions, airlines, and e-commerce is the most severe [24]. Because of this, it is important to prevent failures.

OpenDC uses the notion of a *failure model* to simulate failures, alongside *failure traces*. In OpenDC, a failure constitutes a full host crash, regardless of whether the cause of the failure is a hardware of software problem. A result of a failure in OpenDC, all tasks running on the given host are killed, and need to be rescheduled. A failure model consists of two statistical distributions: (1) to model service unavailability (2) to model service availability. A failure trace is defined by an interval, duration, and intensity of several failures, which are later looped throughout the simulated workload [25]. In summary OpenDC enables experimentation with failures that enables insights that are not provided by other state-of-the-art software. However, the fidelity of failure modeling inside a datacenter simulation is still insufficient to predict in failures in real-time, as they happen in a physical datacenter. Since a datacenter simulator is quite different from a digital twin, we cannot use the same computation methods from simulation to predict real-time failures.

2.4 Datacenter Simulation

Efficient and timely datacenter management is a difficult challenge, because datacenters are extremely complex facilities. They require deep understanding to operate properly. However, running real-world experiments is costly in both time and resources. Additionally, experimentation *in situ* is unsustainable and difficult to reproduce. Alternatives to real-world experiments include simulation and mathematical analysis. Because mathematical analysis is not scalable to modern datacenters [18], in this project we only consider simulation as a foundation for the Datacenter Digital Twin (DCDT).

Simulation empowers better design, testing and management of datacenters [18]. A well-designed datacenter simulator can estimate a months-long workload in a few minutes or hours. To simulate is to "imitate of real-world process or system over time, enabling the study of, and experimentation with the internal interactions of complex systems" [38] In this project we only consider *discrete-event simulation*. Discrete-event simulation repre-

Project	Environment	Stakeholders	Highlighted Features	GUI
CloudSim [26]	Cloud, Fog [27], Edge	Research	VC*, N, S, E, WF† [28], FD†, EXP†, CM, PI†	✓† [29]
SimGrid [30]	Grid, P2P, Cloud [31]	Research, Edu [32]	VC*† [33], N*, S, E*, WF* [34]	✓† [34]
DGSim [35]	Grid	Research	WF, F, EXP	✗
GroudSim [36]	Grid, Cloud	Research	WF, CM, F	✗
iCanCloud [37]	Cloud	Research	VC, N*, S, CM	✓*
OpenDC [18]	Cloud	Research, Edu.	VC*, N, S, E*, CM, FS*, ML, WF, F*, PI, EXP*	✓*

Table 2.1: Comparison of selected datacenter simulators. **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 interface; **Tools:** EXP = Experiment automation; **Support:** ✓ = Yes, ✗ = No; † = extension, not integrated; ⋆ = advanced, carefully calibrated feature. Author: Mastenbroek *et al.* [18]

sents system operations as a sequence of events over time, with an assumption that no changes occur between the events. Due to the scale and complexity of datacenters, most simulators use discrete-event simulation [18]. There exist many datacenter simulation tools, for example DGSim [35], CloudSim [26], SimGrid [30], iCanCloud [37], GroudSim [36] and OpenDC [18]. See Table 2.1 for a comparison of selected datacenter simulators, combined by Mastenbroek *et al.* [18]. In order to narrow the scope of the project, we only consider OpenDC as a simulator for the digital twin design. We decided to use OpenDC, because we find it important for a simulator to model hardware failures well. *Failure models* are a carefully calibrated, advanced feature of OpenDC. Further details about OpenDC can be referred to in the linked literature [18].

2.5 Digital Twinning

In this section we explore how the datacenter management can be improved using a novel modelling technique, digital twinning. We present the generic, field-agnostic DT definition and investigate how *datacenter* digital twinning applies the definition in practice.

2.5.1 What is Digital Twinning?

"A *digital twin* is a set of virtual information constructs that mimics the structure, context and behaviour of a natural, engineered or social system, is dynamically updated with

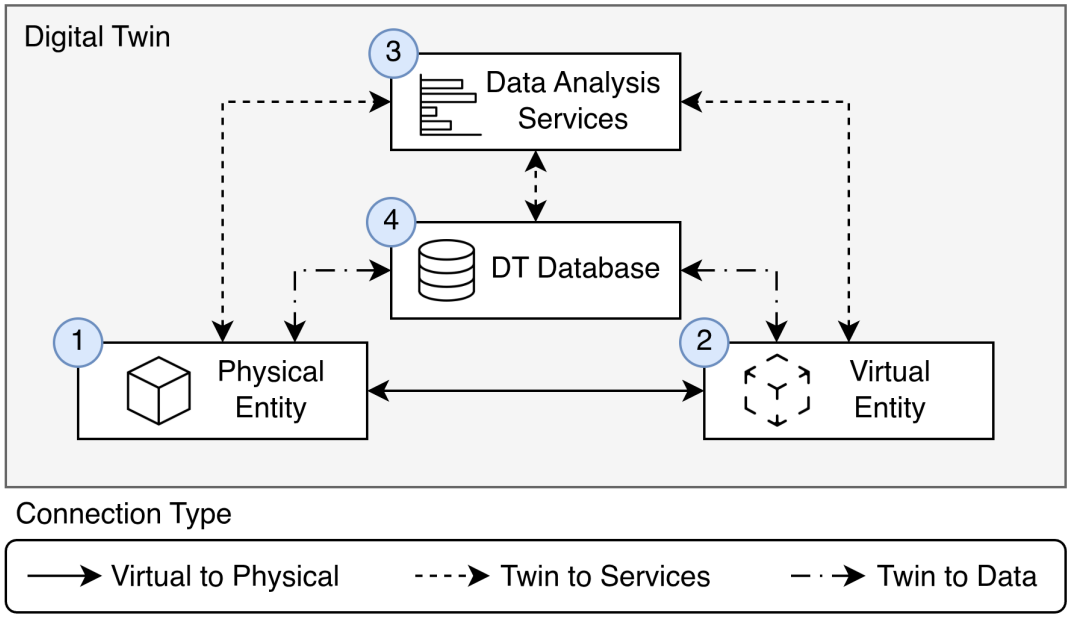


Figure 2.2: A basic framework for the Digital Twin (DT). Four core elements of a DT are defined: The physical entity (❶) and the simulated virtual twin (❷). A service for out-of-band data analytics (❸) and a persistent storage of historical data (❹) are crucial to the DT because they are necessary to gain meaningful monitoring insights. Adapted from Tao *et al.* [8].

data from its physical twin, has predictive capability, and informs decisions that realize value" [39]. A crucial characteristic that differentiates digital twinning from simulation and statistical modelling is the *digital thread*: a bi-directional channel that enables continuous interaction between the virtual and physical entities. The longer the DT is working, the more accurate its predictions, because a holistic twin aggregates historical patterns together with up-to-date monitoring data. A generic DT architecture is depicted in Figure 2.2 from Tao *et al.* [8].

Digital twinning has only recently become feasible because of the developments in High Performance Computing (HPC). Between 2003 and 2011 the compute needed to run a DT was simply not present. As such, while the concept existed, the hardware did not catch up yet. However, in the last decade, multicore computing paradigms and the advent of GPU computing has finally enabled computation needed to run digital twins. As a result, digital twins have become more relevant today than 10 years ago [8].

A crucial part any of any DT is *predictive modelling*, which drives actionable insights [39] (see Figure 2.3). Predictive modelling uses statistics to predict outcomes. When deployed commercially, for example in datacenters, predictive modelling is often referred to as predictive analytics [40]. Almost any statistical model can be used for prediction purposes, but nowadays predictive analysis is synonymous with Machine Learning (ML). A primary

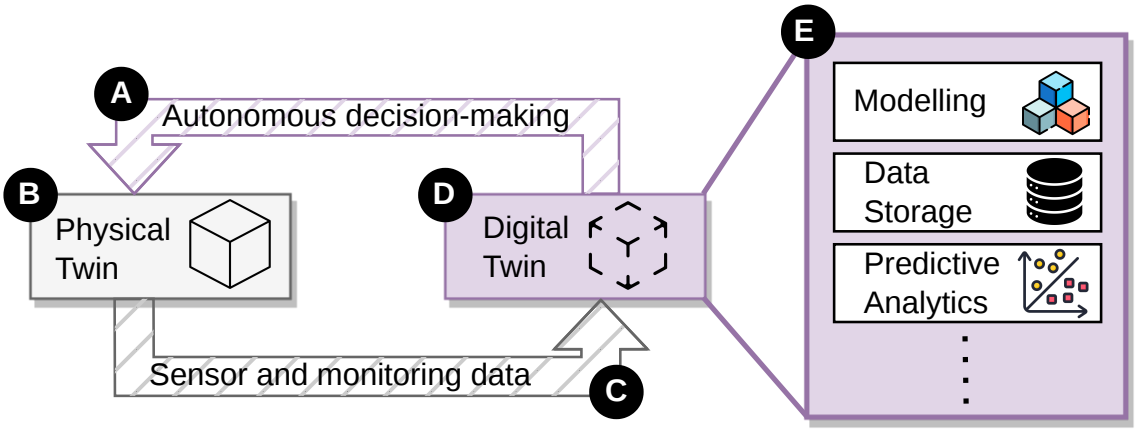


Figure 2.3: Datacenter digital twin diagram. There are 5 core elements to any Digital Twin: Ⓐ The Digital → Physical Twin link, Ⓑ the Physical Twin (*e.g.,* the datacenter), Ⓒ the Physical → Digital Twin link, Ⓓ the Digital Twin, Ⓔ the features necessary to any Digital Twin.

example of popular analysis type is linear regression. However, any modelling technique (*e.g., discrete-event simulation*) can be used to make the predictions.

Predictive analysis belongs to a larger domain of Operational Data Analysis (ODA). ODA is the "use of operational data instrumentation, analysis, integration, and archiving, towards effective design, commissioning, and optimization of datacenter operations" [41].

Operational analytics are present at all layers of a datacenter. A reference architecture, proposed by *Suman et al.* describes the kind of operational analysis performed at each distributed system layer. For example, at the resource manager layer, ODA should enable workload modeling capabilities, which use predictive analysis to determine submitted user job properties [13]. There exist several ODA frameworks, for example OMNI [41], Wintermute DCDB [42], ExaMon [43], AutoDiagn [44] and ODAbler [13].

A major limitation of predictive analytics is that history cannot always predict the future. Using historical data to predict outcomes works only under the assumption that there are certain long lasting patterns in the system. Additionally, no matter how extensive is the training data, there is always the possibility of new variables that have not been considered or even defined, yet are critical to the outcome of the prediction [40].

2.6 Literature Survey of Digital Twins for Datacenters

In this section, we survey the work related to datacenter digital twinning. We summarize our results in Table 2.2 to compare and contrast the features of existing datacenter digital twins. We select only the digital twins that adhere closest to the National Academy of Sciences, Engineering, and Medicine (NASEM) definition [39].

2.6.1 Methodology

The aim of this survey is to search and organize the field of DCDTs. In this subsection, we describe the methods for collecting relevant scientific articles and present the design of the system model for generic DCDTs.

1. **Review Strategy**
   According to Suman *et al.* [13], the most common methods for conducting literature surveys are (1) random traversal of the related literature, (2) snowballing [45], (3) systematic literature survey as proposed by Kitchenham *et al.* [16]. Random traversal encompasses surveying the field by following suggestions from portals like Google Scholar and randomly querying the different databases. It is an unstructured way to conduct the literature review, and requires little effort. Snowballing is similar to random traversal, but it is more structured. The surveyor follows references from the relevant articles, and there is a depth limit [13]. Systematic literature survey is a rigorous, fully-structured process to searching for literature, and it follows the method devised by Barbara Kitchenham [16]. In our work, to scope down the project we chose a mix of (1) and (2), with some elements of (3) instead of following solely the systematic literature review process of Kitchenham *et al.*. Therefore, our literature cannot be regarded as systematic, instead we can refer to it as comprehensive or semi-structured.

2. **Analysis of Selected Material**
   We borrow the process of Suman *et al.* conducted during his MSc thesis for a literature survey of ODA [13]. The process can be described as follows. (1) first, search given queries followed by a manual inspection of the context of the article (2) then scan each article, by reading over the abstract, introduction and conclusion and decide whether it applies to DCDTs. (3) after selection, extract the details of the DCDT from the publication by reading carefully over the article. (4) lastly, interpret the functionality of the DCDTs and systematically organize them.

3. **Design of the System Model**
   Based on the findings of the literature survey, we create a conceptual model of the DCDT field. We decided to create a system model, as the field of DCDTs is still under development, and does not include many digital twin deployments. An alternative to a system model would be a taxonomy. To create the system model, we first gathered the functionality present in all the DCDTs. For each DCDT feature in every article, we evaluated whether this feature is present in other deployments and how important it is for the DT. The result of this process is a set of DCDT features that belong to the largest proportion of all DCDTs. Afterwards, for each of the features, we decide how it is interconnected.

Project	Simulation Technique	Focus	Stakeholders	Modelling Capability
ExaDigiT [12]	CFD/HT, AI/ML	Energy Loss Prediction, Heat Modelling	HPC Engineers and Operators	3D*, CH‡, VP*, PE‡, RA, SE†
SmartDC [46]	CFD/HT, BIM, AI/ML	Heat Modelling, PUE optimization	Cloud Datacenter Engineers	CH†, PE, 3D*
DyTwin [15]	Gaussian Process Regression, AI/ML	Anomaly Detection	Cloud Datacenter Operators	A*, FD, VP*, SE†
ChatTwin [47]	**Unknown**	Digital Twin Definition Language	Cloud Datacenter Engineers	3D*
Reducio [48]	POD, Gaussian Process Modelling (ML)	Heat Modelling	Edge and Hyper-scale Datacenter Operators	CH‡, 3D*, SE
NetGraph [49]	Graphs	Network Management	Cloud Datacenter Operators	VP*, RA*, N*, SE†
Kalibre [50]	CFD/HT, ML	Heat Modelling	Cloud Datacenter Engineers	CH‡, 3D*

Table 2.2: Comparison of selected datacenter digital twins. **Modelling capability:** 3D = Visualizations; CH = Cooling/Heat, PE = Power/Energy Consumption, A = Anomaly Detection, N = Network Modelling, SE = Scenario Exploration, VP = Virtual Prototyping, FD = Federation, RA = Resource Allocation; **Data Analytics**: ‡ = Predictive Analysis; ⋆ = Descriptive Analysis, † = Prescriptive Analysis. **Unknown**: Information is Not Available/Not Published.

2.6.2 Advanced Digital Twins for Datacenters

ExaDigiT [12] is an open-source framework for developing digital twins of supercomputers. It consists of 3 modules: (1) resource allocator and power simulator (2) thermal cooling model (3) augmented reality 3D model of the supercomputer. ExaDigiT has been used at the Frontier supercomputer at the Oak Ridge National Laboratory in the USA, successfully predicting potential energy losses at the supercomputer. Brewer *et al.* include alongside the framework architecture an open-source artifact and a set of extensive verification and validation experiments. The authors differentiate between different digital twins within ExaDigiT, such as (1) descriptive twin (2) informative twin (3) predictive twin (4) comprehensive twin (5) autonomous twin that together form the system. The *predictive twin* leverages data driven operational analytics to create ML models. Authors argue that alongside simulation, ML models should also have a significant role for modeling system workloads in *e.g.,* application fingerprinting. Within the *autonomous twin* the authors use Reinforcement Learning (RL) to train agents that can be used to make control decisions in order to optimize different processes. In order to model the cooling system the authors use the Modelica software, and to predict energy power draw they coded a Python script. The authors provide a intuitive way to interact with the system using a visual dashboard, and an advanced augmented reality model. The authors posit that the best way to address the 3V's of data (velocity, volume and variety) is to use augmented

reality coupled with dashboards.

SmartDC [46] is a digital twin solution for optimization of power consumption in datacenters. Specifically, Zhang *et al.* propose that using Artificial Intelligence (AI) enhanced modeling paired with digital twinning can help make dynamic adjustments to the datacenter cooling subsystem. SmartDC has been proven to ensure efficient energy-saving rate of a China Telecom datacenter at 41%. However, the main purpose of SmartDC is not to continuously interact with the facility, but to provide additional training data for a more accurate, ML solution. The digital twin is designed to provide extra datasets for training AI models.

DyTwin [15] is an adaptive digital twin with visualization and anomaly detection features. The system, developed at Hewlett Packard (HP) is a precursor to the vision on datacenter digital twinning published by Athavale *et al.* [3]. DyTwin is the only system capable of failure detection in datacenters. Moreover, it is the only system to incorporate the idea of federation into the concept of digital twinning. DyTwin is designed to interact not only with the physical facility, but also other federated digital twins. Taheri *et al.* show that DyTwin can effectively detect 100% of CPU usage anomalies (*i.e.,* irregularities that affect CPU utilization, ranging from 5% to 60%).

ChatTwin [47] is an AI and Large Language Models (LLMS) powered system that enables easy deployment and configuration of digital twins for datacenters. It is a *text-to-3D* approach to building digital twins of datacenters. ChatTwin is the only work in the field that does not share the simulation technique used to construct the digital twin based on ChatTwin's configuration. Li *et al.* provide a thorough set of experiments to show ChatTwin generates the JavaScript Object Notation (JSON) DCDT configuration efficiently, but do not share the final 3D visualization results.

Reducio [48] is a system designed to further optimize the Computational Fluid Dynamics (CFD) approach to datacenter modeling. Instead of using plain CFD the authors focus on Proper Orthogonal Decomposition (POD) approaches to approximate the heat transfer. Using the POD technique, the authors are able to model the datacenter more efficiently, achieving sub 1 degree Celsius Mean Absolute Error (MAE) in temperature prediction. Moreover, their model outperforms the CFD approaches. Cao *et al.* evaluate their system on an edge datacenter with 70 CPU-only server racks (see Figure 3 in [48]), and on a hyper-scale datacenter with thousands of servers (see Figure 4 in [48]). Their results show promising gains in physics-based datacenter modelling over the conventional approaches.

NetGraph [49], designed by Huawei Technologies and China Mobile. NetGraph is the only system in our literature review that focuses on network management (see Table 2.2). Moreover, NetGraph employs a unique modelling technique, combining device, network and service models using graph theory. The authors evaluate their system in a Huawei datacenter with over 50000 server racks. With over 20 million connections in the network graphs, the system is a prime example of datacenter digital twin potential.

Kalibre [50] is a system designed by Wang *et al.* in order to overcome the cons of CFD. To lessen the computational overhead, Wang *et al.* propose to use a knowledge-based neural surrogate to calibrate the different CFD models. Kalibre takes the best of both ML

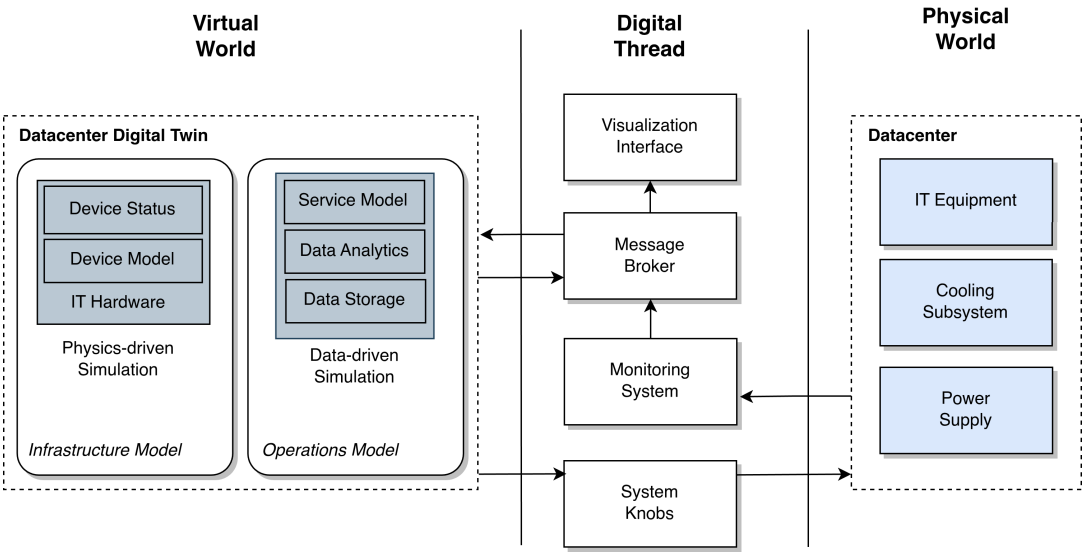


Figure 2.4: A generic system model for datacenter digital twin deployments. The design of DyTwin [15] indirectly incorporates in its architecture a "virtual-to-virtual" digital thread between different digital twins. Zhao *et al.* likewise present key elements to the digital thread in their architecture [51]. We add the *Digital Thread* to our model explicitly.

and CFD approaches and achieves sub 1 degree Celsius MAE, similarly to Reducio [48].

2.6.3 System Model for Datacenter Digital Twinning

In Figure 2.4 we present a holistic model of DCDTs from Section 2.6.2. The figure includes the functionality present in the majority of DCDTs, combined together into a unified model. We distinguish 3 core elements of every DCDT: (1) the virtual world (2) digital thread (3) the physical world.

1. **Virtual World** contains the DCDT. It represents all the components that exist in software. Every DCDT model can be categorized into two sub-categories: (1) infrastructure model (2) operations model. Each DCDT from Section 2.6.2 contains a model of the infrastructure within the datacenter. This includes virtual replicas of the hardware elements (*e.g.,* servers, networking, server racks, rooms). These elements have varying degrees of fidelity. For example, NetGraph models the datacenter interconnect using purely configuration files. On the contrary ExaDigiT models the datacenter hardware fully in 3D. Both offer virtual infrastructure models as a part of the DCDT.

   The operations model is likewise present in all deployments. It models the *behaviour* of the datacenter, *i.e.,* the data flow, the different workloads running on the compute, the amount of data stored in each hosts *etc.*. Both the infrastructure model and the operations model are part of all DCDT deployments from Section 2.6.2. A digital twin that contains only the infrastructure model, cannot enable insights into the real-

time operation of the datacenter. Likewise, a DCDT containing just the operations model does not possess a capability to *e.g.,* simulate the datacenter. Only both, combined together enable the insights envisioned by the NASEM DT definition [39].

2. **Digital Thread** connects the virtual world to the physical world. This is a novel contribution of our thesis. The digital thread is a *conceptual* element that unites the components which do not belong in either of the worlds. All DCDT programs from Section 2.6.2 contain elements that are "in-between" the physical and virtual twin. After comparing and corroborating these components across deployments, we find 4 that prevail the most: (1) the visualization interface (2) the message broker (3) the monitoring system (4) the system knobs. These elements *connect* facilitate the connection between the physical and the virtual. For example, the visualization interface provides insights from the metrics collected by the DCDT (virtual world) to the datacenter operators (physical world). The message broker transfers the data from the real datacenter (physical world) to the digital twin (virtual world).

3. **Physical World** models the real datacenter. All deployments in Section 2.6.2 contain this element. Moreover, within the datacenter, we distinguish between 3 core elements that are necessary to model the datacenter faithfully(1) the IT equipment (2) cooling subsystem (3) power supply. All of the aforementioned systems from Section 2.6.2 model either of the 3 elements. In order to adhere to the holistic view of DCDTs, and to fulfill the NASEM's definition, the system must contain all 3 of these elements.

Chapter 3

Design of *Sunfish*, a Digital Twin For Predictive Analysis of Datacenters

3.1 Overview

Our contribution in this chapter is three-fold:

- C_1 We analyze the requirements for *Sunfish* (Section 3.2).
- C_2 We propose a conceptual design for *Sunfish*'s architecture (Section 3.3)
- C_3 We describe how *Sunfish* fulfills the functional and non-functional requirements in Section 3.4.

3.2 Requirements Analysis

In this section we determine the requirements that should be fulfilled by *Sunfish*. We present here the stakeholders identified by our literature survey (see Section 2.6) and the relevant use-cases. Afterwards, we list the functional and non-functional requirements for *Sunfish*.

3.2.1 Stakeholders

We identify four main stakeholders of a predictive datacenter digital twin:

S1 – Datacenter Managers
Responsible for maintenance and operation of the warehouse, operators manage the datacenter daily. They interact with the servers, bring downed hosts up and ensure customers' services run smoothly at all times. Importantly, datacenter operators follow a strict set of policies for taking different action. Datacenter operators need to ensure different Service Level Agreements (SLA)s are met, energy costs are balanced and carbon emission quota is maintained.

S2 – Datacenter Engineers
The term datacenter engineers encompasses datacenter architects and technicians alike. From the moment datacenter architects determine the warehouse layout, to the physical process of booting the server racks for the first time by the technicians, datacenter engineers help build and maintain the datacenter. They must continuously adapt to changing requirements and ensure everything goes smoothly [3].

S3 – Scientists and Academia
Digital twinning generates unprecedented amount of data. To bring valuable insights, the ingested metrics must be effectively analyzed. Scientists can draw conclusions from the monitoring data and in some deployments already benefit from the voluminous output of Datacenter Digital Twin (DCDT)s [46].

S4 – Customers and Users
Digital twins are already used to visualize both facilities and human beings alike for the benefit of users and customers. Cloud and HPC users are not directly interacting with the system, but pay for services hosted in the datacenter and are one of the primary stakeholders of digital twinning. Not only through 3D datacenter visualizations, but also from the continuously generated metrics can users and customers benefit from DCDTs.

3.2.2 Use-cases

Based on the identified stakeholders we list 6 potential use-cases for a predictive datacenter digital twin:

UC1 – Energy Optimization
Predicting and modeling energy optimization is curcial for DCDTs. It is the main use-case of some existing systems [12]. Effective energy optimization, including the adjustments to the consumed energy type is a crucial use-case of DCDTs.

UC2 – Failure Management
Predictive maintenance of both hardware and software failures alike, by simulating the possible failure distribution of a running workload, can lower downtime, terminated tasks and ensure SLAs are not missed [3].

UC3 – Heat Modelling
Heat modeling is the primary use case of existing DCDTs (see Table 2.2). Correct thermal management of the warehouse can optimize cooling strategies, leading to lower bills and maintenance costs.

UC4 – Network Traffic Modelling
Congestion management and traffic routing can effectively benefit from digital twinning. Detecting bottlenecks, adjusting datacenter protocols and calibrating switches and interconnects is already a important use-case for DCDT [49].

UC5 – Virtual Prototyping

Digital twinning can be used to provide insight into the system before changes are made. Using a DCDT, the operators can change, model and shape the datacenter to estimate their effect. Virtual prototyping encompasses interacting with an existing datacenter model, or with a proof-of-concept, not yet constructed warehouse.

UC6 – Monitoring and Visualization

3D visualizations and dashboards are of the utmost importance to all stakeholders, due to the insights they provide. With real-time data ingestion and the two-way feedback loop, DCDTs can empower descriptive analytics. This use-case already shapes many existing systems (see Table 2.2).

3.2.3 Functional Requirements

Based on a subset of the above use-cases, we formulate the functional and non-functional requirements for *Sunfish*:

FR1 – The system should be able to handle workload size representative of its modelling technique.

Existing Digital Twin (DT)s range from Cloud through the Edge to HPC. Therefore, *Sunfish* must not impose any significant limitations on the size or type of the workload it runs. In particular, *Sunfish* must be able to handle workloads as large as the underlying modelling technique permits. Without **FR1**, *Sunfish* will be incomplete, and like the majority of the Table 2.2 systems, its use-case will be niche. **FR1** is necessary to avoid overly-specializing the DCDT.

FR2 – The system should support failure detection.

Failures are of crucial concern to datacenter operators. The system detect and report failures to datacenter operators. Without **FR2**, the system will not be able to help datacenter operators meet SLAs. Including **FR2** is necessary to ensure that failures, which which can cause financial penalties, are detected in a timely manner so as to meet the different SLAs.

FR3 – The system should be capable of long-term and short-term data storage.

FR3 is necessary for **FR4** and **FR5**. Without **FR3**, the system cannot support Operational Data Analysis (ODA) techniques. A system cannot be considered a DT without insights stemming from accurate data analytics [39]. National Academy of Sciences, Engineering, and Medicine (NASEM) digital twin definition requires both real-time insights, and guidance based on historical-patterns. Therefore, it is imperative to ensure **FR3**.

FR4 – The system should support descriptive data analytics.

There are many types of data analytics present is existing deployments [13]. In order to scope down the project, we only select several use-cases for *Sunfish* (SF). Out of

the 4 prime analytics type, we find in the literature survey the predictive analytics to be the most urgently needed, and descriptive analytics to be the most prevailing type of data processing. Therefore, in order to achieve performance on par with previous systems, and to ensure our system meets the official NASEM definition, **FR4** is needed.

FR5 – The system should enable predictive data analytics.
The system should help future researchers incorporate predictive data analytics engines, regardless of the statistical modelling technique. This is our novel contribution to the scientific field of DCDTs. Without **FR5**, we miss the aim of our entire project.

FR6 – The system should be able to process arbitrary amounts of telemetry.
The size of the Artificial Intelligence (AI) economy is expected to grow [3]. As a result, both current and future datacenters will generate huge amount of data. Our system must be capable of ingesting and processing different metrics regardless of the volume of incoming messages. Without **FR6**, we hinder the adoption of *Sunfish* in modern and future datacenters.

FR7 – The model should be capable of clear data visualization. The system must have a user-friendly, visual interface for data analytics, and support them in real-time. Without **FR7**, we exclude important stakeholders such as customers and users from the potential benefactors of our system.

3.2.4 Non-functional Requirements

In addition to the functional requirements, we also present non-functional requirements for *Sunfish*:

NFR1 – Using *Sunfish* should not introduce any delays in visualizations longer than 1 second.
The system must work in real-time, without significant delay. We impose 1 second delay as acceptable to datacenter engineers. We have arrived at the 1 second threshold due to the fact that datacenter operators need time to react to sudden changes in the facility behaviour. To expect a sub-1-second performance from human employees is unjustified. At the same time, any delay > 1 second can already negatively impact engineer's ability to react to different events at the same time in the correct order (*i.e.,* > 1 delay hinders the ability of technicians to follow the specific datacenter protocols and policies during an event). The system must support datacenter operators with insights at fine-grained granularity, so that insights derived from data analysis remain accurate upon reception by datacenter operators. Without **NFR1**, *Sunfish*'s insights will not be timely, and will be useless to datacenter operators.

NFR2 – The system should log the ingestion and processing of metrics.
The system should provide a log of the current network traffic to and from the digital twin.

NFR3 – The system should adhere to modern software development standards. The system should follow modern coding principles and guidelines to ensure reproducibility and usefulness for future work. Without **NFR3**, it will be difficult for current and future digital twin developers to include predictive analytics using *Sunfish* in their deployments.

NFR4 – The system should provide insights at varying levels of confidence. With the huge amount of telemetry data incoming from the physical twin, the digital counterpart must be able to filter and pre-process the telemetry. Without **NFR4**, the system will present overwhelming amount of information to its users, rendering it unusable.

3.3 Overview of *Sunfish* Architecture

As a result of the *AtLarge Design Process* [17] designed a reference architecture for a predictive datacenter digital twin. Figure 3.1 encompasses 4 main elements: (I) physical datacenter (II) digital thread (III) digital twin (IV) predictive analytics.

3.3.1 The Physical Datacenter

The Physical Datacenter (I) encompasses 3 core elements important to digital twinning. Workloads (1a) include the hardware requirements of each datacenter job and the submission time. They are executed on the datacenter compute (1b), which is controlled partly by the Datacenter Operators (1c). Component (1c), while seemingly unimportant, is crucial to the digital twin design. We envision DCDTs as systems that contain a human-in-the-loop, which can control and overwrite the system's autonomous decisions. Datacenter Operators (1c) interact with both the Servers (1b), and have the ability to overwrite the potential autonomous digital twin decisions, stemming from component (2c), the System Knobs.

3.3.2 The Digital Thread

The Digital Thread (II) is a novel contribution from Chapter 2. It is a crucial element of our architecture. It separates the physical world from the virtual world, and contains components that do not belong to either twin, or belong to both twins. Otherwise, it would be unclear how to model the data flow between the physical and the digital twin (*i.e.,* which elements take part in the information exchange). Conceptually separating the physical and virtual twin makes it easier to comprehend and compartmentalize the design. It contains the Interactive Dashboard (2a), the Message Broker (2b), and System Knobs (2c).

To fulfill the functional requirements of our system, we incorporate element (2a), the Interactive Dashboard to our system. This dashboard ingests data coming directly from the Message Broker (2b) and from the long-term storage (3a), the Database. The Interactive

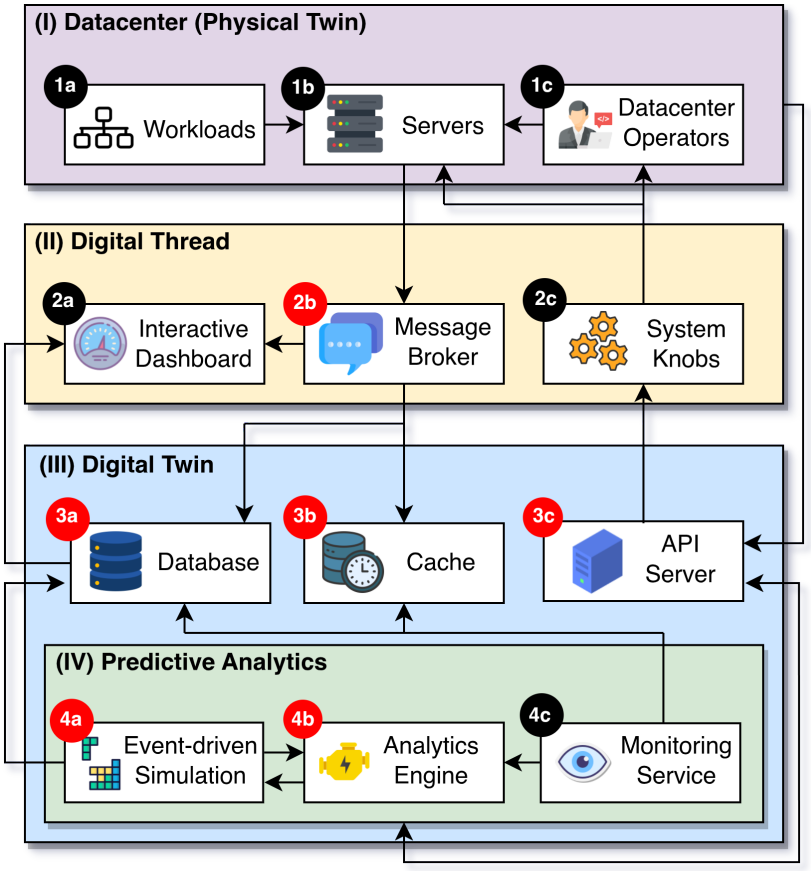


Figure 3.1: The predictive datacenter digital twin reference architecture. We call the system *Sunfish*. The architecture was designed with the *AtLarge Design Process* [17] over several iterations in the past months.

Dashboard (2a) allows datacenter operators to see the telemetry data arrive to the digital twin in real time.

The Message Broker (2b) is a crucial component to the reference architecture, because it facilities the physical twin → virtual twin connection. A low-latency, high-throughput message broker partly meets our functional requirements to enable arbitrary amounts of telemetry data transfer. The Message Broker (2b) is a component is slightly more complex, and necessities a separate diagram. In Figure 3.2 we present the composite elements that make up the Message Broker. In particular, the *schema registry* and the *connector manager* play a crucial part in fulfilling the functional requirements of our system. The *schema registry* allows the telemetry producer to submit *any* data format for sending (and storing) the telemetry data.The registry is responsible for detecting what kind of format is the data sent in, and automatically adjusting the schema within the *data pipeline*. The connector manager is responsible for joining multiple distinct services to a single data pipeline. In the reference architecture, the consumers would constitute the Database

(3a), the Cache (3b), and the Interactive Dashboard (2a). What is remarkable about the connector manager is the ability to swiftly connect more consumers to the system. This way, the predictive digital twin can facilitate multiple different types of analytics engines or techniques. Additionally, the setup in Figure 3.2 is currently standard industry practice for large software deployments.

The System Knobs (2c) represent the different cogs within the datacenter software and hardware (1b) that can be adjusted during runtime (*e.g.,* to optimize Power Usage Effectiveness (PUE), change cooling strategy, allocate compute resources). For example, System Knobs (2c) within the datacenter scheduler can be tuned to schedule jobs on Servers (1b) that are least likely to experience future downtime. The autonomous actions of the digital twin (the tuning of the System Knobs (2c)) can be further adjusted by Datacenter Operators (1c).

3.3.3 The Digital Twin

In our design, we explicitly differentiate between the physical and virtual space by including the Digital Twin (III) in a separate box. The Digital Twin (III) constitutes of the long-term storage (3a), short-term storage (3b), the API Server (3c) and the Predictive Analytics (IV) module.

Long-term storage, namely a Database (3a) is crucial to enable real-time visualizations (2a), and to model long-term system behaviour. This fulfills the functional requirements for our system, as we set out to differentiate between in-band data analytics, and out-of-band data analysis. The data is consumed by the Database (3a) directly from the Message Broker (2b). In-between, simple data pre-processing can take place.

Short-term storage, in the form of a Cache (3b) is essential for in-band data analytics, "on-the-go". The Cache (3b) retains the data consumed from the Message Broker (2b), and for a short period of time keeps a copy. The data analytics performed on the Database (3a) dataset and the Cache (3b) adhere to the use-cases for predictive (long-term) and descriptive (long-term, short-term) data analytics.

The Message Broker (2b) enables one-way connection (physical twin → digital twin). The API Server (3c) enables the digital twin → link. The API Server (3c) communicates directly with the System Knobs (2c) in order to take meaningful action, based on the (predictive) insights generated by the Digital Twin (III). Additionally, the Physical Twin (III) can query the API Server (3c) for one-shot requests (*e.g.,* to create a new datacenter prototype configuration, to request special data analysis). Moreover, the Datacenter Operators (1c) can query the API Server (3c) for extra insights, when necessary.

3.3.4 The Predictive Analytics Engine

The Predictive Analytics (IV) module is an extensible part of the reference architecture, enabling different kinds of predictive analysis. In our design, to facilitate meaningful predictions we incorporate an Event-driven Simulator (4a), Analytics Engine (4b), and a Monitoring Service (4c).

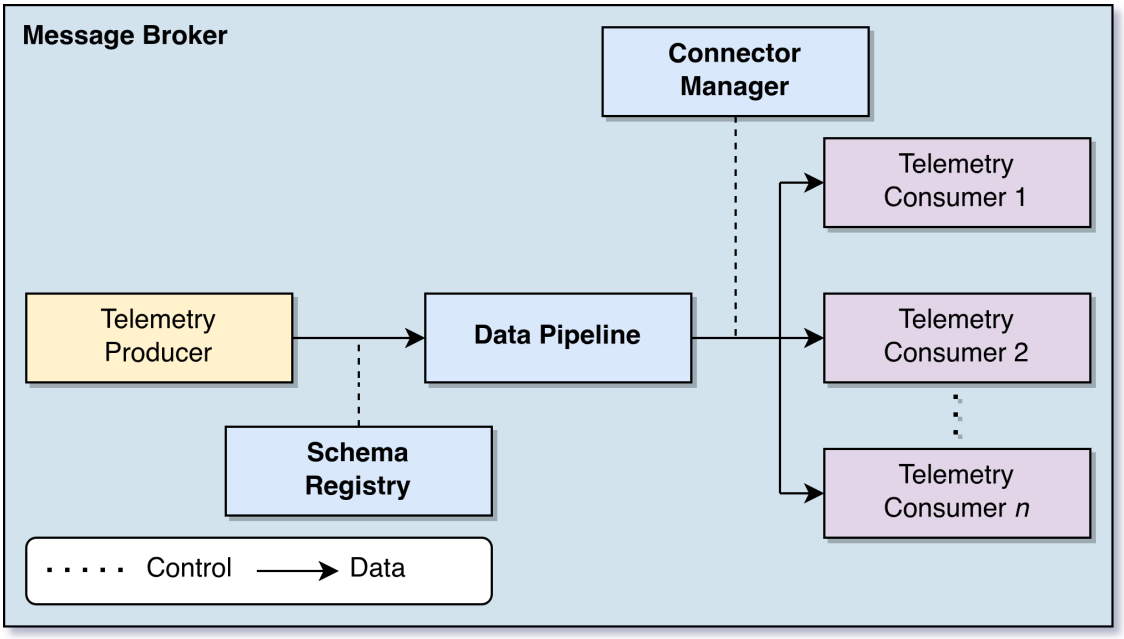


Figure 3.2: The detailed view of the Message Broker (2b) from Figure 3.1.

We chose an Event-driven Simulator (4a) as a core element of predictive analytics. For the rationale behind this decision see Section 2.2. The predictions from the Event-driven Simulator (4a) communicate directly with the Database (3a) to store the simulation results, and with the Analytics Engine (4b).

The Analytics Engine (4b) performs descriptive and predictive data analytics on the results from the Event-driven Simulator (4a) and the telemetry data collected from the physical datacenter (stored in (3a, 3b)). This component is highly extensible to accommodate the different types of analytics (*e.g.,* Machine Learning (ML), AI, statistical methods). The results of the data analysis are propagated to the API Server (3c) to be queued for retrieval by the physical twin.

The Monitoring Service (4c) serves as a separate thread to watch over the collected telemetry in real-time. It facilities detection of irregularities in collected data, which adheres to the functional requirements set out for the system. Any discrepancies are communicated to the Analytics Engine (4b) for further analysis, and potential insights.

3.4 Requirement Validation

In this section we describe the rationale behind the different design decisions of *Sunfish*. To show that our model satisfies the functional and non-functional requirements, we map each component of *Sunfish* onto the requirements it fulfills.

The Interactive Dashboard (2a) supports (**FR2**), as it allows to display the failures detected to datacenter operators; it is the core of (**FR4**) because descriptive analytics are

	FR1	FR2	FR3	FR4	FR5	FR6	FR7	NFR1	NFR2	NFR3	NFR4
Interactive Dashboard (2a)		✓		✓	✓		✓	✓	✓	✓	
Message Broker (2b)	✓	✓	✓	✓	✓	✓		✓	✓	✓	
System Knobs (2c)		✓		✓	✓					✓	✓
Database (3a)			✓	✓	✓	✓			✓		
API Server (3c)		✓		✓	✓					✓	
Cache (3b)			✓	✓	✓			✓	✓	✓	
Event-driven Simulation (4a)	✓	✓		✓	✓			✓		✓	✓
Analytics Engine (4b)	✓	✓		✓	✓						✓
Monitoring Service (4c)	✓	✓		✓	✓						

Table 3.1: Validation of the features against the set of functional and non-functional requirements.

synonymous to visualizations. It enables (**FR5**) because the predictive insights can be displayed directly through the dashboard to the datacenter operators. Operators can then overwrite the autonomous decisions performed by the DT. Dashboards are the current industry practice for clear data visualization (**FR7**). In our design, we ensure (**NFR1**) is in a two-fold way. Firstly, we ensure a direct connection between the Interactive Dashboard (2a) and the Message Broker (2b). As the Message Broker (2b) is capable of accommodating multiple consumers, the Interactive Dashboard (2b) receives the same telemetry at the same rate as the live prediction engine. Secondly, through a direct connection to the Database (3a), the Interactive Dashboard (2a) can instantly retrieve all historical patterns. Thus, we meet (**NFR1**). (**NFR2**) and (**NFR3**) are met by including in the system implementation a dashboard that enables logging of metrics.

The Message Broker (2b) indirectly fulfills (**FR1**). A good and robust message broker alleviates potential bottlenecks that might stem from large data transfer. My including a dedicated message broker, we ensure the bottle neck when twinning is not *Sunfish*. As the Message Broker (2b) is a vital component in the communication between the datacenter operators and the physical twin, it indirectly enables (**FR2**), (**FR3**), (**FR4**), (**FR5**). In the other direction, to ensure that the DCDT can ingest any amount of data, the Message Broker (2b) is needed for (**FR6**). The non-functional requirements are met by including a state-of-the-art message broker from either academia or the industry in the implementation.

The inclusion of System Knobs (2c) in the reference architecture fulfills (**FR2**). With autonomous actions, the DT can alter the operation of the datacenter based on the detected failures. Failure detection must support datacenter operators in meeting SLAs. Autonomous actions, enabled by System Knobs (2c) help achieve that. (**FR4**) and (**FR5**) are met by ensuring the insights that come from the DT can be automated with the system knobs. Supporting both predictive and descriptive analytics is core to the system knobs. (**NFR4**) is met as the existing digital twin deployments often contain (indirectly) the System Knobs (2c) element. Including it in our design adheres to best design practice. Importantly, (**NFR5**) is fulfilled together by creating a predictive analytics engine that is

capable of intelligent insights and system knobs. In order for the predictive insights to be realized, systems knobs are needed.

The Database (2c) meets (**FR3**), (**FR4**), (**FR5**), (**FR6**) and (**NFR2**). Trivially, including a simple Database Management System (DBMS) in the architecture fulfills (**FR3**). For the different types of analytics (**FR4**), (**FR5**), the Database (2c) enables insights from historical patterns. Database are the current state-of-the-art in storing telemetry (**FR6**). They are designed to hold arbitrary volume of data. Lastly, (**NFR2**) is met by storing in the database the logs from both the DCDT and the physical datacenter.

The API Server (3c) ensures (**FR4**), (**FR5**), (**FR2**) and (**NFR3**). The server, which facilitates the insights that come into the System Knobs (2c), by proxy enables (**FR4**) and (**FR5**). To detect failures, some sort of notification daemon is needed for the datacenter operators, not just the dashboard. The API Server (3c) fulfills that role, ensuring (**FR2**). The API server (3c), specifically a server using the Hyper-text Transfer Protocol (HTTP) protocol is a state-of-the-art approach to existing digital twin communication (**NFR3**).

The Cache (3b) ensures the same requirements as the Database (3a), with a couple of exceptions. Namely, it also ensures (**NFR1**) and (**NFR3**). Caching is needed for sub 1-second visualizations, and the inclusion of a cache for in-band data analytics is the current community practice.

In our design, we include an Event-driven Simulator (4a). This ensures (**FR1**), (**FR2**), (**FR4**), (**FR5**) and all non-functional requirements except (**NFR2**). Using a discrete-event simulator ensures that our system can handle workloads that are representative of the simulator at use. Simulation, a method that is widely accepted by the scientific community, is one way *Sunfish* can ensure (**FR1**). Detecting failures, enabling both descriptive and predictive analytics depends on how good of a simulator (4a) is. In principle, a robust, holistic simulator incorporated into the predictive engine is capable of meeting (**FR4**), (**FR2**) and (**FR5**). Simulation also allows to model large workloads and to interact with the simulation in real-time (**NFR1**), it is considered a state-of-the-art approach to datacenter modelling (**NFR3**) and, with good data analytics provides enough data for insights at varying levels of confidence (**NFR4**).

The Analytics Engine component (4b) and the Monitoring Service (4c) fulfill the same set of requirements, namely (**FR1**), (**FR2**), (**FR4**) and (**FR5**). Additionally the Analytics Engine (4b) also fulfills (**NFR4**), as it provides the capability to differentiate between different levels of confidence. Both components are working together in unison, closely watching and reacting to the metrics from the real datacenter. Hence, they fulfill (**FR2**), (**FR4**) and (**FR5**) and are crucial to the DCDT operation. (**FR1**) is fulfilled by ensuring both the Analytics Engine and the Monitoring Service are not the source of any bottlenecks during the system implementation.

Chapter 4

Implementation of *Sunfish*

4.1 Overview

The contribution of this chapter is two-fold:

C_1 We implement the real-world prototype of *Sunfish* (see Section 4.2) realizing key features of the design.

C_2 We engineer *Sunfish* to enable predictive analytics engines (see Section 4.6)

4.2 Detailed Implementation of *Sunfish*

In this section we describe the detailed implementation of *Sunfish* (SF). After reading one should understand the technical decisions, choice of tools and modifications to existing software necessary for evaluation of SF in Chapter 5.

Any complex system is more than the sum of its parts [22]. To understand SF it is crucial to provide a holistic view on the prototype. Therefore, the rest of this chapter is structured in a top-down approach: Section 4.2 presents the rationale for using the specific software packages, Section 4.3 shows the flow of data within the system, and Section 4.5 details the different modifications and new software extensions to `OpenDC`. Lastly, Section 4.6 carefully explains the design decisions behind the major Python modules.

At the onset of the project, we decided SF will use only state-of-the-art software, deployed in the industry or evaluated in peer-reviewed scientific publications. The mapping of software packages used onto the reference architecture can be seen in Figure 4.1. In order to facilitate visualizations and interactive dashboards, we decided to use `Grafana` (2a) [52]. To enable the flow of data into the Digital Twin (DT), we use `Kafka` (2b) [53]. To store the in-band data we use a `Redis` (3b) [54] cache, and for out-of-band data we use a `PostgreSQL`(3a) [55]. To enable predictive analytics, we chose a discrete-event simulator, `OpenDC`(4a) [25]. The `Analytics Engine` (4b), `Monitoring Service` (4c), and `HTTP Server` (3c) are described in detail in Section 4.6.

`Grafana` (2a) is a state-of-the-art industry tool to visualize dashboards. We posit it is cru-

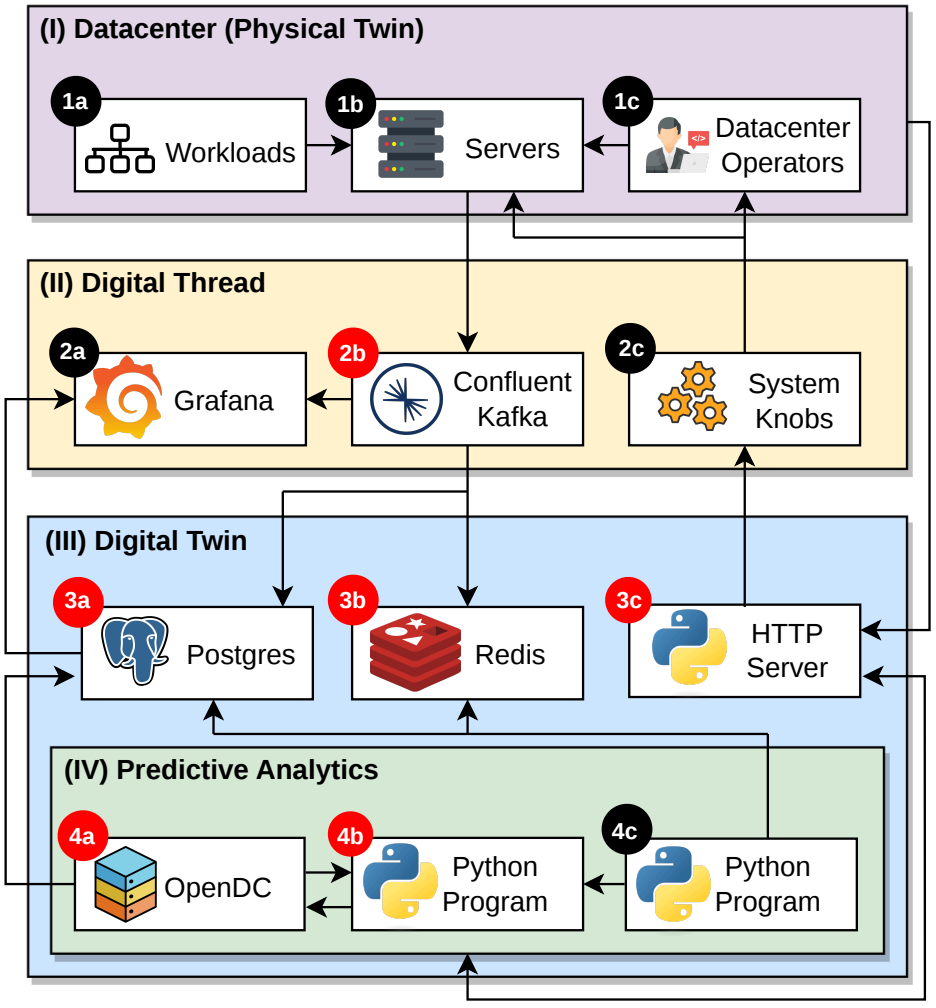


Figure 4.1: The prototype and its components based on the architecture. The time-series data flows first to the `Grafana` (2a) dashboard, `PostgreSQL` (3a) database and `Redis` (3b) cache as advised in [15].

cial to include a user-friendly performance dashboard. A number of previous publications on DTs find dashboards important [15, 13, 56]. We chose `Grafana` (2a) instead of other software packages because of its seamless integration with `PostgreSQL` (3a). `Grafana` (2a) provides good separation of concerns and compartmentalization as it does not store the displayed metrics itself. Instead, it queries the `PostgreSQL` (3a) database in real-time [52], unlike *e.g.,* `InfluxDB`. Good alternatives to `Grafana` (2a) are `Kibana` [57], `Prometheus` [58], and `Graphite` [59].

`Kafka` (2b), particularly `Kafka` developed by Confluent [53] is a battle-tested message broker that provides versatile capabilities to transfer huge volumes of data with little latency, in real-time. We decided to use `Confluent Kafka` instead of `Kafka` developed by the Apache Foundation, because of it's masterful connector system allowing to easily

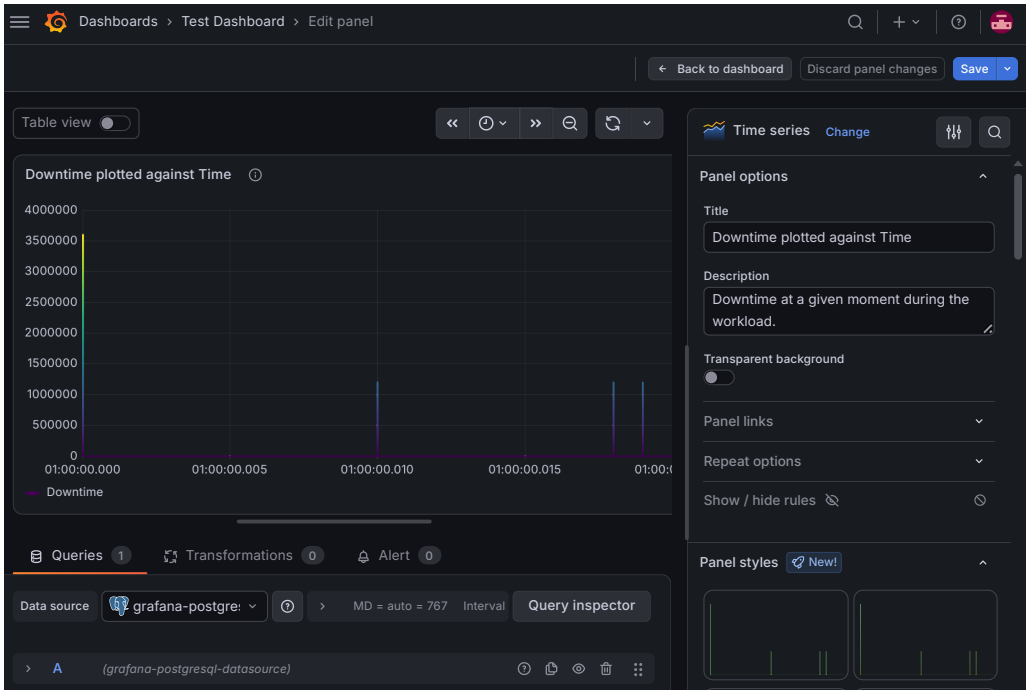


Figure 4.2: Example Grafana Dashboard with downtime plotted against time.

add sources and sinks (*e.g.*, PostgreSQL (3a) sink, Redis (3b) sink, OpenDC source (4a)). Additionally, as opposed to Apache Kafka, Confluent Kafka comes equipped with a Schema Registry. The Schema Registry is a important component that allows the storage of database and cache schemas for easy retrieval. With Schema Registry, we ensure that the data stored in PostgreSQL (3a) tables and in Redis (3b) streams contains the exact same schema. Moreover, Schema Registry is compatible with versatile data interchange formats, such as ProtoBuf [60] (see Listing 4.1).

Redis (3b), is a key value store that provides efficient store and retrieval operations [54]. In particular, Redis (3b) is capable of storing *streams* – append only logs which allow for fast and quick query of large volumes of data. Redis (3b) is the industry leader in key value caching. The only alternative to Redis (3b) is memcached [61], which does not provide the capability to integrate with Kafka (2b).

PostgreSQL (3a) is a database management system, necessary to store large volumes of out-of-band data coming from the physical datacenter. The PostgreSQL (3a) server provides a simple and straightforward interface to query the data via psql. Importantly, to adhere to the single responsibility principle, PostgreSQL (3a) does not provide any User Interface (UI). Additionally, there exist many integrations between PostgreSQL (3a) and other software, including Kafka (2b). The many alternatives to PostgreSQL (3a) are listed in [55]. An alternative used in previous work is InfluxDB [13].

Lastly, to enable predictive analytics we use a state-of-the-art discrete-event simulator, OpenDC(4a) [25]. OpenDC (4a) is a leading software package capable of modeling complex

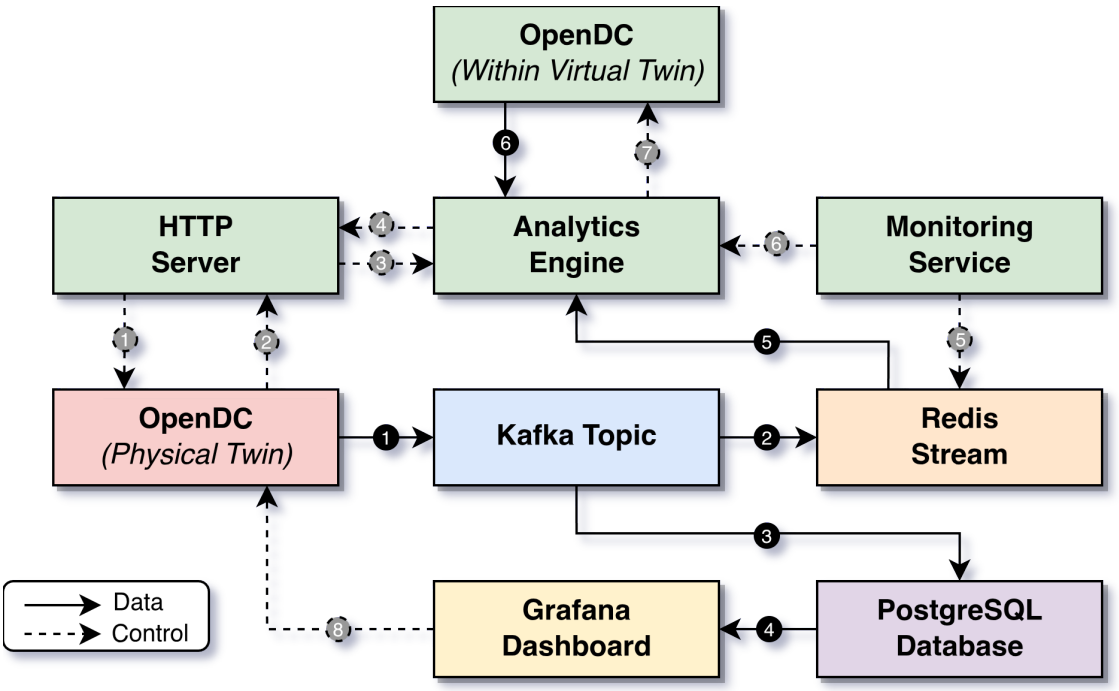


Figure 4.3: The data flow within *Sunfish*.

datacenter phenomena and workloads (*e.g.,* failures, workflows, machine learning). For a specific overview of advantages of `OpenDC` (4a) and a thorough comparison with other alternatives, see Table 2.1.

4.3 Data Flow

In this section we describe the data flow within Figure 4.1 using a separate diagram. Efficient data flow is of utmost importance to DTs. In Figure 4.3 we present the moving of data within SF. In the diagram whenever we refer to *control*, we mean small, one-in-a-while data packets that contain either instructions, insights or small amount of data. Whenever we refer to *data*, we mean the hundreds of thousands of metrics exported with minimal latency from the operating datacenter (`OpenDC`). We describe the flow of data through a timeline.

First, the datacenter informs the DT of an upcoming workload (2). This packet contains the datacenter topology and the upcoming workload tasks (workload trace, collected from *e.g.,* BitBrains). The DT stores this data locally, and passes it forward to the `Analytics Engine` (3). The analytics engine queries the `OpenDC` simulator to run a simulation of what might happen in the datacenter under such workload (7). `OpenDC` returns the potential results to the `Analytics Engine` directly (6). This data for the purposes of the prototype is stored in the `.parquet` files. In real-world scenario, this data flow (6) would be connected to a separate `Kafka` topic.

```
syntax = "proto3";

package proto;

option java_package = "org.opendc.common";
option java_outer_classname = "ProtobufMetrics";

message ProtoExport {
  string timestamp = 1;
  string host_id = 2;
  int32 tasksactive = 3;
  double cpuutilization = 4;
  double energyusage = 5;
  double uptime = 6;
  double downtime = 7;
}
```

Listing 4.1: The Redis and PostgreSQL schema (`schema.proto` file). All the large volume data packets that travel within the system follow this format (*i.e.,* ❶ through ❻). The communication of small datapackets takes place using the Hyper-text Transfer Protocol (HTTP) protocol.

In the meantime, the datacenter executes the workload. The datacenter continuously sends the metrics into the `Kafka` topic (❶) (*i.e.,* the datacenter is the *producer*). As this happens in real-time, `Redis` ingests the data from the `Kafka` topic (❷) alongside `PostgreSQL` (❸) (*i.e.,* `Redis` and `PostgreSQL` are the *consumers*). At the same time, `Grafana` polls the `PostgreSQL` database for incoming metrics (❹). Each time `PostgreSQL` ingests the data from the `Kafka` topic (❸), at the same time `Grafana` updates its real-time dashboard (⑧). This provides real-time feedback to datacenter operators (⑧).

Simultaneously, the `Monitoring Service` checks the in-band data within `Redis`, in real-time (❺). Should anything unusual occur, the `Monitoring Service` notifies the `Analytics Engine` to perform necessary analysis (❻). Then, the `Analytics Engine`, ingests the data from the `Redis` stream (❺) and analyzes it for further insights. All insights generated in this way, are sent to the `HTTP Server` (❹) to communicate to the system knobs within the datacenter and to the datacenter operators (①).

Of significant importance are data flows (❷) and (❸). Due to the massive volume of data incoming from the physical datacenter, the `Redis` sink (❷) has been configured to filter out relevant packets. `Kafka` comes with excellent capability to efficiently compare data packets against a condition and filter our packets that are of no use to the `Analytics Engine` (see Listing 4.3). On the contrary, the `PostgreSQL` sink (❸) contains all metrics collected by the datacenter sensors (see Listing 4.2). This setup achieves excellent abstraction level, because only the most important metrics are forwarded to the `Analytics Engine`, with the majority of packets being filtered out.

```
name=postgresql-sink
connector.class=io.confluent.connect.
    jdbc.JdbcSinkConnector
tasks.max=1
# Crucial for Schema Registry
key.converter=org.apache.kafka.
    connect.storage.StringConverter
value.converter=io.confluent.connect.
    protobuf.ProtobufConverter
value.converter.schema.registry.url=
    http://localhost:8081
topics=postgres_topic
connection.url=jdbc:postgresql
    ://127.0.0.1:5432/opendc
connection.user=matt
connection.password=admin
auto.create=true
auto.evolve=true
insert.mode=insert
table.name.format=postgres_topic
```

Listing 4.2: Kafka PostgreSQL sink.

```
name=kafka-connect-redis
topics=postgres_topic
tasks.max=1
connector.class=com.redis.kafka.
    connect.RedisSinkConnector
key.converter=org.apache.kafka.
    connect.storage.StringConverter
value.converter=io.confluent.connect.
    protobuf.ProtobufConverter
value.converter.schema.registry.url=
    http://localhost:8081
transforms=downtime
transforms.downtime.type=io.confluent
    .connect.transforms.Filter$Value
transforms.downtime.filter.condition=
    $[?(@.downtime == '0.0')]
transforms.downtime.filter.type=
    exclude
transforms.downtime.missing.or.null.
    behavior=fail
```

Listing 4.3: Kafka Redis sink.

4.4 The OpenDC Scheduling Paradigm

In this section we introduce the scheduling paradigm of OpenDC. OpenDC, a robust datacenter simulator uses discrete-event simulation. "Discrete-event simulation models the operation of a system as a (discrete) sequence of events in time" [62]. Colloquially, it is akin to calling an `Update()` method on a set of objects to model changes in the simulator. Scheduling in OpenDC also works using this method. Figure 4.4 represents how a task is assigned to a host in the simulation.

A task (❶) is represented by its submission time, duration and computational requirements. In order to be assigned a server to run on, it is deserialized into a `ServiceTask` (❷). The `FilterScheduler` (❹) class takes care of scheduling the service task once the simulation reaches its submission time. OpenDC maps tasks to available hosts via an *allocation policy* [63]. In our work, this is always (also in `SmartScheduler`) the `FilterPolicy`. In the `FilterPolicy`, a series of `HostFilters` (❸) and `Host Weighters` (❺) are used to find the matching host for the task. In this project, the different filters and weighters take into account the default metrics (*i.e.,* Central Processing Unit (CPU) capacity, Random Access Memory (RAM) capacity, number of CPU cores). The host itself is represented as a `HostView` class (❻). Importantly, the `HostView` class does not serve to simulate the behaviour of the host, but to provide a interface for the *current state* of the host. For actual computation, the `SimHost` (❽) class is used. The `SimHost` object is created via the `HostProvisioningStep` class (❼) [63].

4.5 Extensions to OpenDC

OpenDC is a state-of-the-art datacenter simulator. In order to turn it into a DT, we have made several design decisions and extensions.

1. **SmartScheduler**
   The new SmartScheduler is a scheduling mechanism capable of incorporating the insights from the DT into its scheduling decisions. It relies on the functionality of the HTTPClient to poll the DT at each scheduling step for potential insights. For example, if DT sends to the datacenter a list of hosts likely to fail in the future, the SmartScheduler acts as *system knobs* to enforce the DT insights (*i.e.,* it can be mapped to ❷c from Figure 4.1). It *replaces* the FilterScheduler (❹) in Figure 4.4. Importantly, it is functionally almost exactly the same as the FilterScheduler, with the exception that it can change the scheduling outcome based on the information from the DT. It is a FilterScheduler with an attached network socket.

2. **KafkaMonitor**
   The datacenter acts as the *producer* of metrics, ingested by the Kafka topic (see Figure 4.3). We equip OpenDC with a new ComputeMonitor capable of exporting data directly into a Kafka topic. The KafkaMonitor class implements the generic ComputeMonitor interface, therefore the new extension follows the other metric exporting options in OpenDC (*i.e.,* follows the implementation for exporting metrics into .parquet files).

3. **HTTPClient**
   The HTTPClient offers the necessary functionality to communicate between the DT and the datacenter. We decided to use the HTTP protocol for short, one-off communications between the DT and the datacenter, as is common industry practice.

4. **CpuUtilVictimSelector**
   In order to ensure OpenDC can model different failure injection algorithms, we added new methods for selecting the hosts to be stopped. The CpuUtilVictimSelector selects hosts based on their current CPU utilization (*i.e.,* the hosts with the highest utilization are stopped first).

5. **RoundRobinVictimSelector**
   The RoundRobinVictimSelector is another algorithm to used to inject failures into hosts. This method works in a round robin fashion, using the alphabetical order of the hosts (*i.e.,* by name). For example, in a scenario hosts H-01 to H-20 are stopped first, and then H-20 to H-40 are stopped next, *etc.*

6. **RandomVictimSelector**
   This is the default OpenDC victim selection algorithm. It randomly chooses which host to stop when injecting failures. This component is left unmodified, but we include it here for the purposes of listing all the failure injection algorithms we use together.

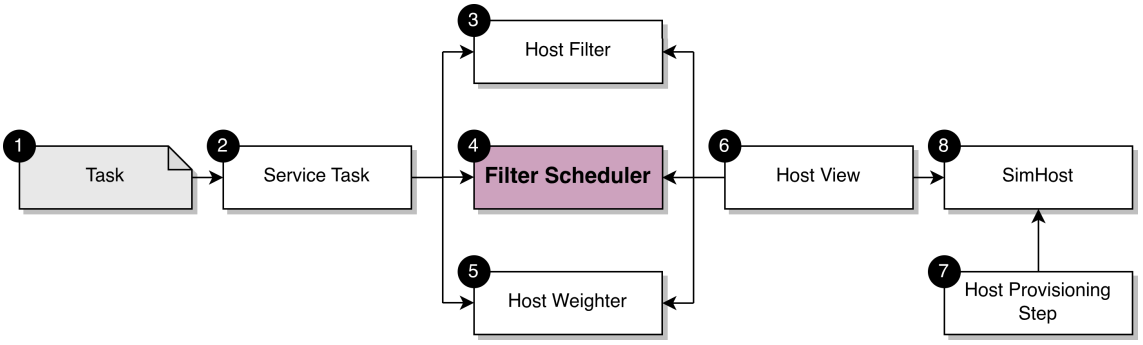


Figure 4.4: The scheduling paradigm in `OpenDC`. Adapted from Muscă *et al.* [63]. The highlighted ■ **Filter Scheduler** is the component that is substituted by the `SmartScheduler`.

4.6 Python Modules

`Analytics Engine`, `HTTPServer`, `MonitoringService` are Python modules we prototyped for the reference architecture evaluation. It is important to note that we do not evaluate the performance of SF due to the substantial overheads of the Python interpreter. `Kafka`, `Redis` and `PostgreSQL` enable milisecond-latency and huge throughput within the system, however what stops us from a performance evaluation is the high cost of Python interpretation. For future work, we envision a system that implements the reference architecture in a compiled language, *e.g.,* `C` or `C++`.

1. **`AnalyticsEngine`**
   The `AnalyticsEngine` module is necessary in order to encapsulate the logic of data preprocessing and analysis from monitoring. This component can contain capabilities for different statistical metrics, subject to DTs focus. In SF `AnalyticsEngine` continuously checks whether the incoming datacenter sensor readings exceed different thresholds. For example, the `AnalyticsEngine` is capable of calculating a similarity score S between potential failure distributions and the true failure distribution.

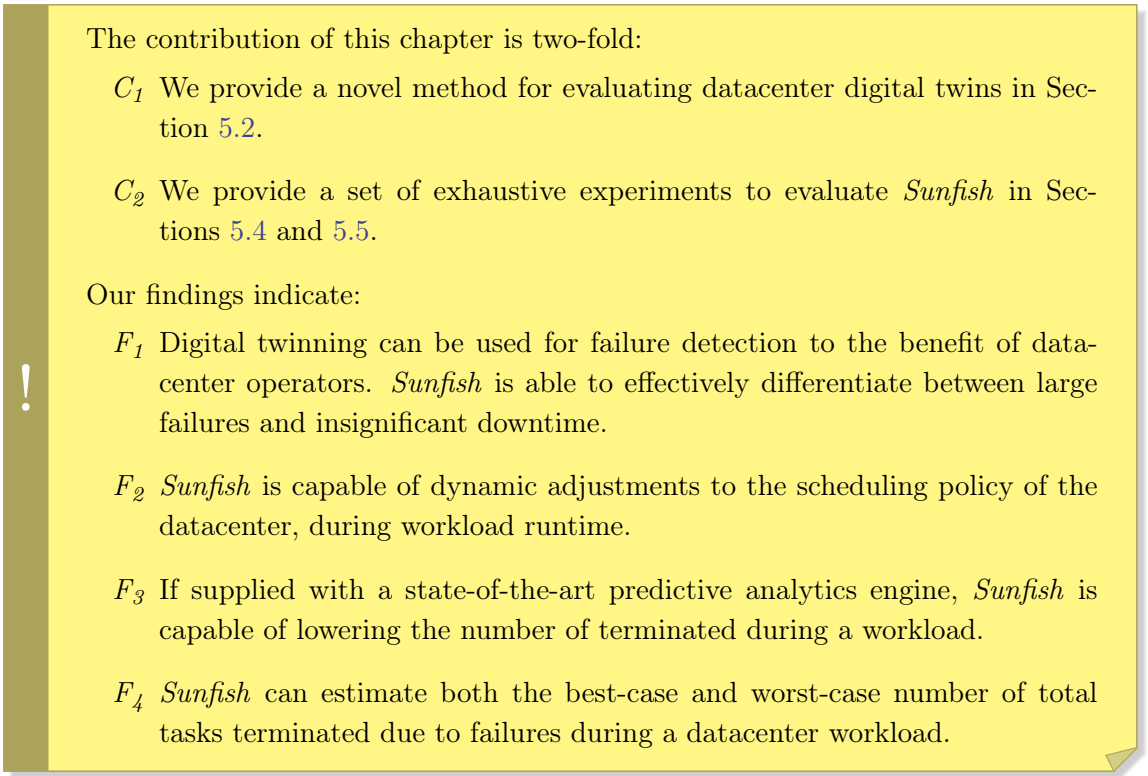
2. **`HTTPServer`**
   The `HTTPServer` is crucial for interrupting the operation of the datacenter in order to adjust its operation or offer insights. It maintains a python `Queue` structure. The `Queue` *producer* is the `AnalyticsEngine` (④). The *consumer* is the `HTTPClient` within `OpenDC` (*i.e.,* the real datacenter, (❷), (❶)). Both the consumer and producer interact with the `Queue` via the network and different HTTP endpoints of `HTTPServer`.

3. **`MonitoringService`**
   The `MonitoringService` is responsible for interacting with the `Redis` key value store. It contains a `while True` Python loop which contains the function call to fetch the latest changes to the `Redis` stream. Upon update, the `MonitoringService` informs the `AnalyticsEngine` that new data is awaiting `AnalyticsEngine` (⑥).

Chapter 5

Experimental Evaluation through a Real-World Prototype

5.1 Overview



The contribution of this chapter is two-fold:

- C_1 We provide a novel method for evaluating datacenter digital twins in Section 5.2.
- C_2 We provide a set of exhaustive experiments to evaluate *Sunfish* in Sections 5.4 and 5.5.

Our findings indicate:

- F_1 Digital twinning can be used for failure detection to the benefit of datacenter operators. *Sunfish* is able to effectively differentiate between large failures and insignificant downtime.
- F_2 *Sunfish* is capable of dynamic adjustments to the scheduling policy of the datacenter, during workload runtime.
- F_3 If supplied with a state-of-the-art predictive analytics engine, *Sunfish* is capable of lowering the number of terminated during a workload.
- F_4 *Sunfish* can estimate both the best-case and worst-case number of total tasks terminated due to failures during a datacenter workload.

5.2 Novel Evaluation Technique

In the this section, we evaluate *Sunfish* using `OpenDC` and all the necessary software (*i.e.,* `Grafana`, `PostgreSQL`, `Confluent Kafka`. A naïve experimental approach would consist of connecting *Sunfish* to a physical datacenter. However, as explained earlier in this work,

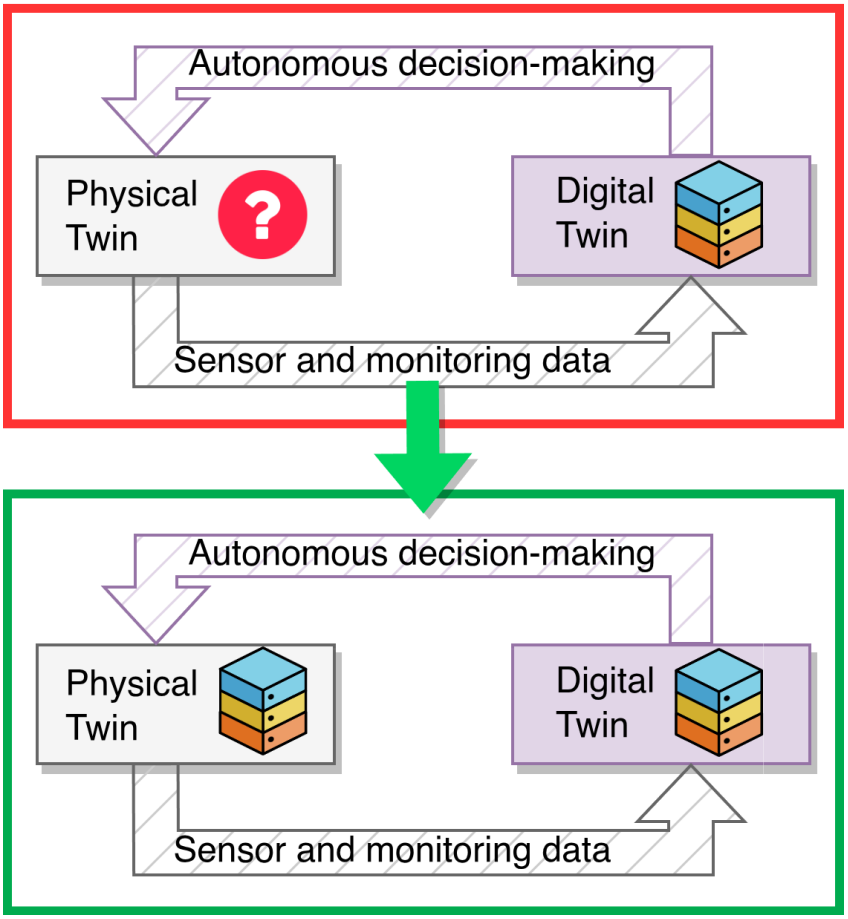


Figure 5.1: A novel evaluation method which solves the issue of real-world experimentation, which is unsustainable and costly [18]. In this approach, both the physical and virtual datacenter are emulated using discrete-event simulation.

running large experiments on real-world data centers is expensive and time-consuming [18]. Additionally, we do not have access to a datacenter. To alleviate this problem, we again utilize simulation for the benefit of evaluating *Sunfish*, and Datacenter Digital Twin (DCDT)s in general.

In this non-trivial approach, we replace the real-world datacenter with *another* instance of the event-driven simulator from Figure 3.1. For our implementation, we decided to use a second `OpenDC` process. A schematic overview of this experimental technique can be seen in Figure 5.1, where the ■ **red** box highlights our initial approach, and the ■ **green** box shows the novel evaluation method. The advantages of this approach are as follows: (1) The "physical twin" simulator is capable of fully replacing the real-world facility, and allows for reproducible experimentation. (2) The environmental footprint is smaller. As a result, *Sunfish* is more sustainable. (3) Datacenter engineers can fine-tune the digital twin using this setup before pairing it with the production environment. In result, engineers can effectively use our setup to test and debug the digital twin before installation. (4) With

Trace	Mean Failure Intensity
Gmail	53.26%
WhatsApp	57.97%
YouTube	62.1%
Twitter	65%
Facebook	64%

Table 5.1: This table shows the traces we used for scientific evaluation, and the right-hand-side shows the format of failure traces. Mean Failure Intensity can be characterized as the average of the ratio of hosts that are affected by a failure [65]

this method, DCDT research becomes more accessible both to the scientific community and higher education.

5.3 Experiment Parameters

In this section we detail the experimental setup that follows most of the experiments. The workload trace used for all experiments is based on a one-month SURF workload trace [64]. In the experiments we model a Dutch SURF datacenter for scientific computing. The cluster, SURF-SARA, contains 277 hosts, each with 128GB of RAM and 16 processing cores running at maximum 2.1GHz [56]. This setup ensures a realistic experiment scenario that is representative of a real-world setting.

The `OpenDC` that models the physical datacenter uses the `SmartScheduler`. The event driven simulator within the predictive analytics engine of *Sunfish* uses the `FilterScheduler` by default, unless otherwise specified.

For the experiments that model datacenter compute failures, we use *failure traces* or *failure models*. For a detailed explanation on the differences between the two, consult Section 2.3. In short, failure traces contain the data necessary to reproduce the outages experienced by the real datacenter during the simulation. A trace contains 3 elements, failure interval, failure duration and failure intensity (see Table 5.1). A failure model is characterized as as a statistical distribution of failures based on scientific research. It is akin to the output of an algorithm that determines when failures should happen, rather than a real-world trace [23]. In the experiments we use traces from the archive developed by Talluri *et al.* [65]. We chose a diverse range of failure traces, based on the mean failure intensity in each trace (see Table 5.1). Mean failure intensity is a metric that shows best how severe the failures are. The higher the failure intensity, the more hosts go down at once, resulting in more serious downtime and missed Service Level Agreements (SLA)s. According to the mean failure intensity, we chose the 5 traces in Table 5.1 for our experiments.

In Section 5.5 we used a failure trace from Skype. This is the only trace that can be

	Availability intervals				Unavailability intervals			
Trace	**Exp**(μ)	**Wbl**(k, λ)	**LogN**(μ, σ)	**Gam**(k, λ)	**Exp**(μ)	**Wbl**(k, λ)	**LogN**(μ, σ)	**Gam**(k, λ)
lanl05	1779.99	048 816.60	5.56 2.39	0.35 5102.71	5.92	0.58 2.18	0.05 1.42	0.38 15.44
g5k06	31.41	0.48 14.37	1.51 2.42	0.34 94.35	7.41	0.35 0.47	-2.00 2.20	0.19 39.92
microsoft99	67.01	0.55 35.30	2.62 1.84	0.41 162.19	16.49	0.60 9.34	1.42 1.54	0.46 35.52
websites02	11.85	0.46 3.68	0.23 2.02	0.31 38.67	1.18	0.65 0.61	-1.12 1.13	0.50 2.37
pl05	159.49	0.33 19.35	1.44 2.86	0.20 788.03	49.61	0.36 5.59	0.40 2.45	0.21 237.65
ldns04	141.06	0.51 79.30	3.25 2.33	0.39 362.43	8.61	0.63 5.62	0.91 1.64	0.51 16.87
overnet03	2.29	0.85 2.04	0.19 0.98	0.91 2.53	12.00	0.44 2.98	0.08 1.80	0.29 41.64
nd07cpu	13.73	0.45 4.16	0.30 2.20	0.30 46.16	4.25	0.51 0.74	-1.02 1.27	0.28 15.07
skype06	16.27	0.64 10.86	1.60 1.57	0.53 30.79	14.31	0.63 9.48	1.40 1.73	0.50 28.53

Table 5.2: The failure models table, by Javadi *et al.* [23].

Metric	**Datatype**	**Unit**	**Summary**
Failure Interval	int64	miliseconds	The duration since the last failure.
Failure Duration	int64	miliseconds	The duration of the failure.
Failure Intensity	float64	ratio	The ratio of hosts affected by the failure.

Table 5.3: All traces used in our work follow the above format.

paired with a corresponding failure model (see Table 5.2). Additionally, in Section 5.4 we find a need to define a threshold based on a statistical distribution of failures. For this purpose, we use a normal distribution with mean 1.5 and standard deviation 1.5.

All the experiments were run on a commodity laptop, with an AMD Ryzen 7840U CPU containing 16, double-threaded cores and maximum frequency of 5.13 GHz, 32GB of DDR5 RAM. By showing that *Sunfish* works using this setup, we encourage scientists and academia (stakeholder **S3**) to freely experiment with *Sunfish*.

5.4 Experiment 1: Failure Detection

For the first experiment we copy the idea of Taheri *et al.* [15]. We show 2 different ways a DCDT can notify datacenter engineers of detected failures.

5.4.1 Context

In this section we try to provide the rationale behind this experiment. Failure detection is a primary use-case for DCDTs [3, 39, 15, 9]. Failures in a datacenter can arise from a number of problems, *e.g.,* software configuration issues, power outage, network congestion [6], and must be immediately detected to ensure their negative impact is minimized. Some failures can have negligible consequences (around 38% of all failures are negligible [6]), whilst other

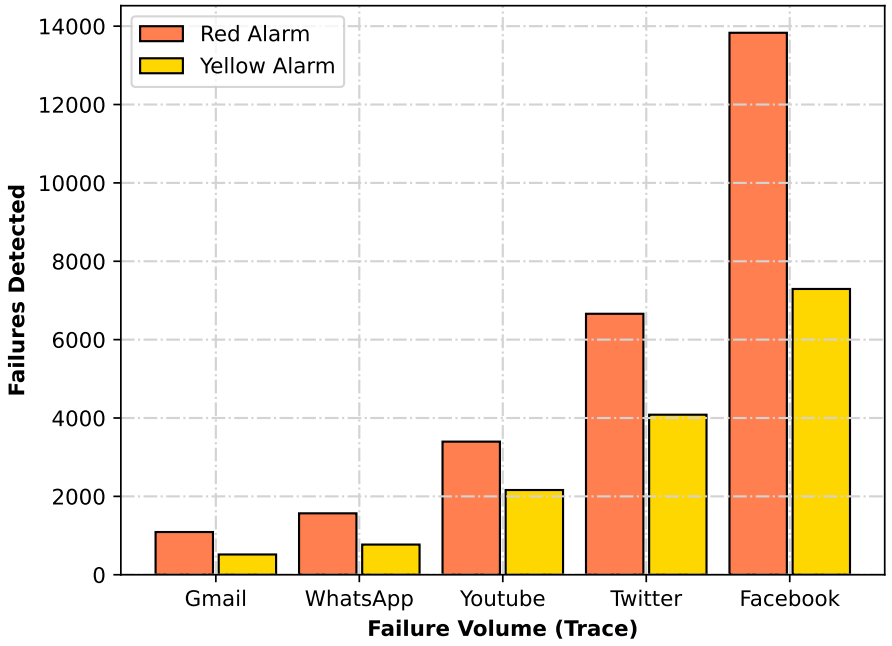


Figure 5.2: The results of Experiment 1. ■ Red Alarms signify 90% of acceptable failure threshold was reached. ■ Yellow Alarms signify 80% of the threshold was reached.

can cause severe downtime and loss of millions of $USD [6]. By differentiating between insignificant and severe failures, datacenter operators can focus their efforts on addressing the most impactful problems, minimizing missed SLA and ensuring swift datacenter operation.

In Section 5.4 we try to model such scenario. For example, imagine the following sequence of events:

1. The datacenter receives a scheduled workload.

2. To predict what kind of failures might occur during the workload we ask the digital twin to run the workload first.

3. We have no *a priori* knowledge of the workload type, so we can only estimate the distribution of failures. To do this, the DCDT will run the workload, assuming failures follow *e.g.,* a normal distribution.

4. After running the simulation, the DCDT can analyze the results and compare them with the running workload.

5. If the running workload has more failures than what the DCDT predicted, it notifies the datacenter operators that something is wrong. Through the real-time feedback loop, the DCDT notifies the datacenter operators that what is happening in reality is different from simulation.

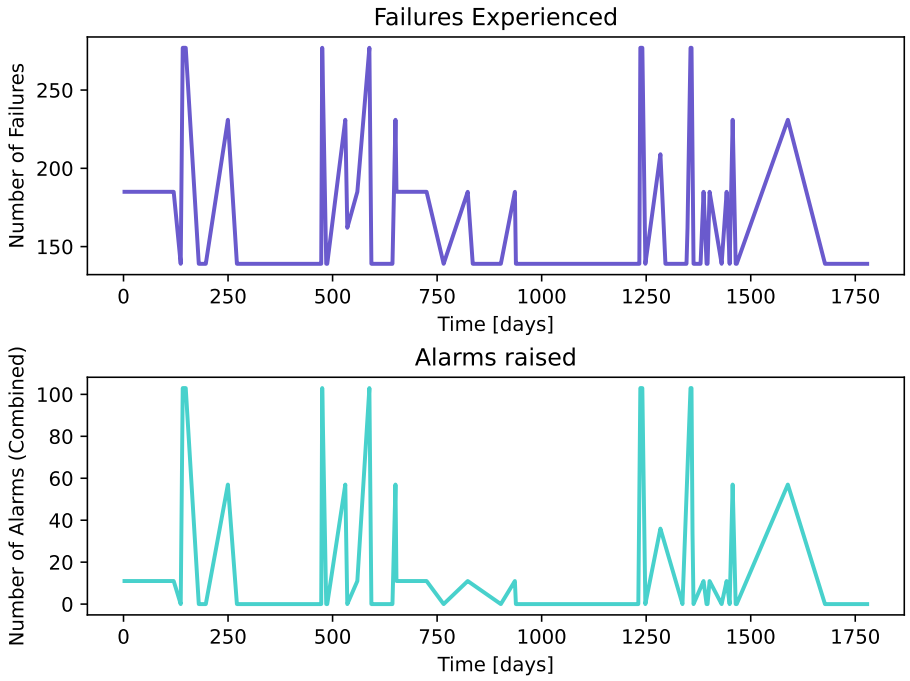


Figure 5.3: Comparison between the total number of raised alarms and the ground truth failure distribution during a SURF month workload in the SURF-SARA cluster. The failure trace used in this plot models Gmail outage reports [65].

In our experiment, we notify the datacenter operators in two cases: if we get within 80% of the predicted threshold for number of failures we send a yellow alarm. If we get within 90% we send a red alarm.

The purpose of this experiment is two fold: (1) to show our system works correctly (2) to show our system fulfills the functional and non-functional requirements. The technical experimental setup can be described as follows: (1) firstly, we use `OpenDC` and a failure model with the normal distribution $\mathcal{N}(\mu = 1.5, \sigma = 1.5)$ to model the failures we might expect from a given workload. (2) then, using the predictions, we establish a threshold acceptable to datacenter operators (*i.e.,* how many failures can we tolerate before we raise any alarm) (3) the red alarm is raised when 90% of the threshold is reached, and the yellow alarm is raised when 80% of the threshold is reached. (4) lastly, the `OpenDC` acting as the real datacenter runs the workload, and *Sunfish* closely monitors the datacenter to see if the number of failures exceeds the accepted threshold. The results are in Figures 5.2 to 5.4.

5.4.2 Discussion

Figure 5.2 indicates *Sunfish* is capable of accurately detecting failures in datacenters. In the figure we can see that as the mean failure intensity rises, so does the total number of failures detected. This validates our design, as this behaviour is expected – the higher the mean failure intensity, the more severe the failures, the more notifications to datacenter

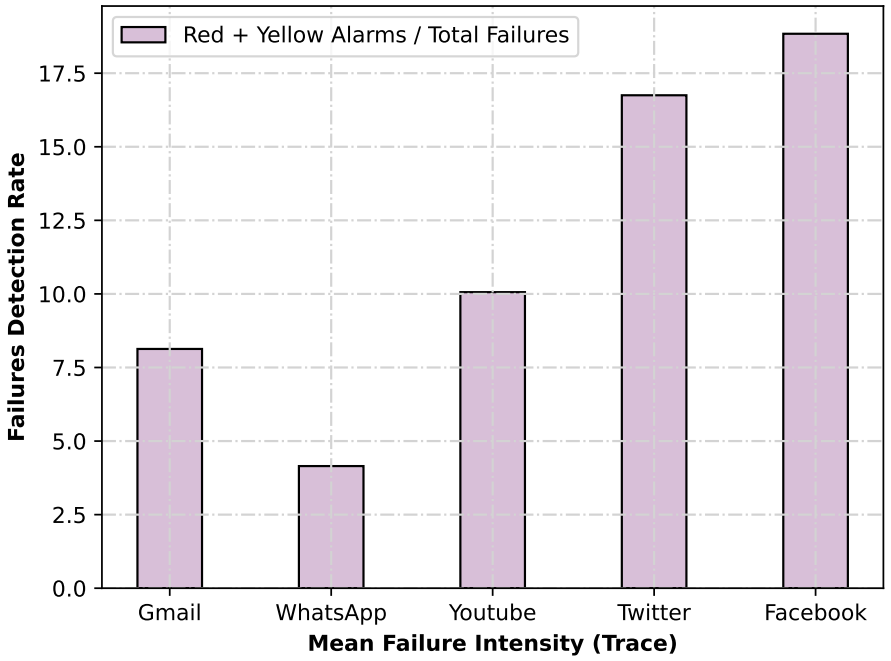


Figure 5.4: In this figure we show the total failure detection rate (■ Red + Yellow Alarms / Total Failures). Our results are much different from DyTwin's performance [15]. We believe this is due to the irreconcilable differences between our experimental setups.

operators are sent. The more failure-intense the trace, the more alarms are raised on behalf of the digital twin. What is more, using the different threshold values, *Sunfish* can differentiate between serious failures and insignificant, single host problems (*i.e.,* the number of yellow alarms is smaller than the number of red alarms). To offer a second perspective, we provide Figure 5.3. In this figure, we show the total number of alarms raised and the ground truth (how many failures occurred in reality.) We can see a clear correlation between the total number of alarms raised, and the actual number of failures have occurred at each time during the workload. For this visualization, we combined both the red and yellow alarms into a single metric. In short, Figure 5.3 backs our claims, and verifies the results obtained in Figure 5.2.

However, Taheri *et al.* present their results differently, using the *anomaly detection rate* instead. The rate is simply calculated as the anomalies detected correctly over the true amount of anomalies [15]. Therefore, in Figure 5.4 we also plot the failure detection rate. What is surprising is Taheri *et al.* report almost negligible false positive rate and of their system. Moreover, they conclude through DyTwin's experimental setup, Taheri *et al.* achieve 100% anomaly detection rate. In our experiment, the numbers differ significantly. Figure 5.4 shows the mean failure detection rate to be around 12%. Compared to the DyTwin deployment, the difference is staggering. We stipulate the discrepancy stems from the fact in our setup we differentiate between different types of failures. This capability is not present in the DyTwin digital twin [15]. As a result, we interpret Figure 5.4 as showing

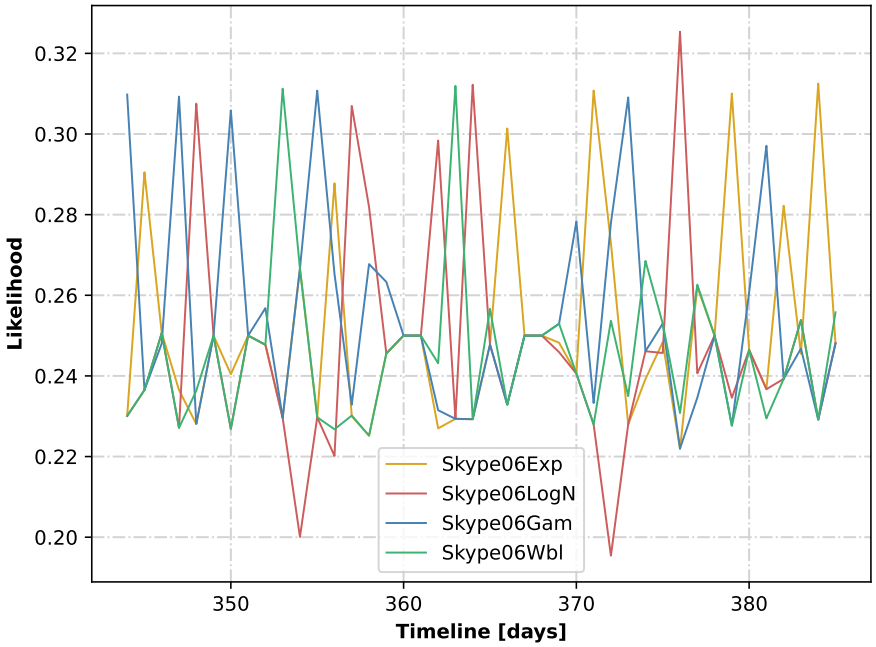


Figure 5.5: How closely a *failure model* approximates the real failure distribution (the Skype failure trace) during runtime.

.

that on average 12% of failures in the workload are severe. Surprisingly, the WhatsApp failure detection rate is the lowest, contrary to the mean failure intensity, which places WhatsApp trace as the 2nd least failure-intensive trace.

5.5 Experiment 2: Failure Prediction

In Section 5.4 we show *Sunfish* is capable of incorporating descriptive analytics. Through experiment-based evaluation, we concluded *Sunfish* can detect and differentiate between severe and one-off host failures. In this section we try to show *Sunfish* can additionally work together with a predictive analytics engine, enabling actionable insights into the future behaviour of the datacenter.

5.5.1 Context

In order to predict when a host failure might occur, the most straightforward approach is to use long-established statistical methods. Our goal was to approximate the real failure distribution of a workload, using past data and relevant statistical distributions. For the task at hand, we chose the Skype Failure Trace, because it is supported by 4 different failure models, based on past Skype workload data. These 4 statistical distribution, published in a peer-reviewed journal are in Table 5.2 [23]. The Skype Failure Trace model was taken

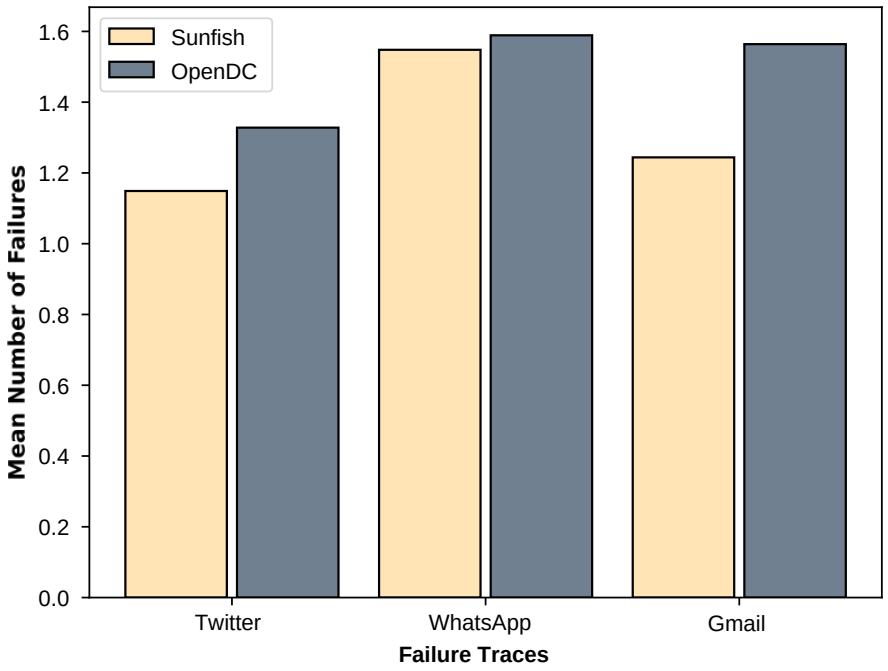


Figure 5.6: The conceptual experiment. ■ Sunfish performance shows the benefits of digital twinning over a normal datacenter without digital twinning insights ■ OpenDC. Lowering the mean number of failures is of significant importance to datacenter operators.

from the Cloud Uptime Archive [65]. The goal is to use the failure distribution to predict when a host will fail, and then in advance re-schedule all the tasks from the hosts onto different machines *before* it crashes.

Initial experiment results were unpromising. Using the insights from the failure models we were not able to do better than the baseline (switching hosts on and off randomly). To investigate why this might be the case, we run an experiment to identify which failure distribution at any given moment is most likely to resemble the actual, ground truth failure distribution. Using a similarity score $\mathcal{S}$, which is a weighted average of the exported metrics, we tried to determine the most similar distribution at any given time. The results are shown in Figure 5.5.

In Figure 5.5 we can notice an almost random fluctuation of the similarity score $\mathcal{S}$. Any given failure model, at any time interval is almost as likely to model the actual failures as the other models. Moreover, the similarity score $\mathcal{S}$ of each failure model is never higher than 32%. Figure 5.5 exacerbates the difficulty of predictive analytics.

Despite the aforementioned results, we set out for a different solution to show *Sunfish* is capable of incorporating a predictive analytics engine. Instead, we designed a *conceptual experiment*. In this setup, we *assume* the predictive analytics engine is capable of fully predicting when each failure is going to happen with 100% accuracy. Equipped with this assumption, which only serves to show *Sunfish* meets the functional and non-functional

requirements, we conducted the second experiment. The results are shown in Figure 5.6.

5.5.2 Discussion

Figure 5.5 shows that using a perfect predictive analytics engine, *Sunfish* is capable of lowering the total number of failures significantly. In this experiment we used only 3 failure traces from user reports of Twitter, WhatsApp and Gmail. Importantly, we assume in this experiment a perfect precognition of what failures might occur and when. While unrealistic in a practical scenario, this assumption serves to approximate the potential gains from incorporating a predictive analytics engine with *Sunfish*. The results are only indicative. In the figure we can see that the mean number of failures in a "physical datacenter" (*i.e.,* `OpenDC`) is much higher without the insights of *Sunfish*.

5.6 Experiment 3: Failure Exploration

For this experiment, we adapt the idea of multi-modal simulation from Nicolae *et al.* [66] to estimating failures with DCDT.

5.6.1 Context

In this experiment we use the DCDT to explore the number of terminated tasks when different failure injection algorithms are used. The primary objective of this experiment is to identify the worst-case scenario and best-case scenario of running a workload prone to failures.

Nicolae *et al.* explore how multi-modal simulation can provide additional insight into datacenter operation and can potentially halve the error of singular models [66]. For our work, we adopt a similar approach to failure prediction. Imagine the following sequence of events:

1. The datacenter receives a scheduled workload, and the operators want to know the worst-case scenario and best-case scenario of how severe failures they can expect.

2. Datacenter engineers ask the digital twin to run the workload first with different failure injection algorithms.

3. After running the simulation, the DCDT can provide the maximum and minimum number of expected failures.

4. By continuously monitoring the datacenter during the runtime of the real workload, the operators can use the estimate to make more informative decisions.

For this experiment we used a SURF month trace running on the SURF-SARA cluster, as explained in Section 5.3. Additionally, we use the Gmail failure trace to inject failures, and use 3 different algorithms for selecting the hosts to fail: `CpuUtilSelector`, `RoundRobinSelector`, `RandomVictimSelector`. The results are depicted in Figure 5.7.

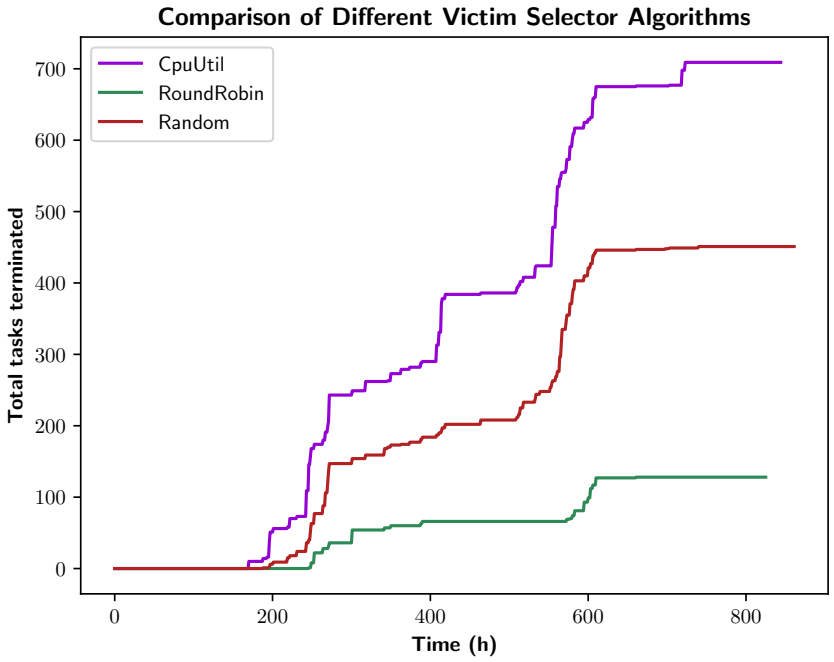


Figure 5.7: Comparison of different algorithms used for injecting failures into the simulated digital twin to explore the worst-case and best-case scenarios.

5.6.2 Discussion

Figure 5.7 shows using different victim selection algorithms can effectively present the worst-case and best-case number of tasks terminated before running the real workload. The victim selector that achieves the most tasks terminated is the `CpuUtilSelector`, which stops the hosts with the highest Central Processing Unit (CPU) utilization. It reaches over 700 terminated tasks (709) in total. The `Random` selector, which randomly picks the hosts to stop, achieves the second-highest number of tasks terminated, at around 450. The most optimistic scenario for datacenter operators is presented by the `RoundRobin` selector, which picks the hosts to terminate in a round-robin fashion. No more than 130 tasks are terminated using this fault injection algorithm. To summarize, this experiment shows that *Sunfish* can be used for the exploration of different failure algorithms for the benefit of datacenter operators. Using *Sunfish*, on-site engineers can effectively estimate the best-case and worst-case number of terminated tasks during a workload.

Chapter 6

Conclusion

Datacenter manageability is a top-priority for the digital society. Over 3 million jobs in the Netherlands directly depend on cloud services, which are hosted in datacenters [1]. Datacenter digital twinning, a promising management technique can offer unique insight into complex warehouse behaviour [3]. In this thesis we paved the way to more advanced Datacenter Digital Twin (DCDT)s. We contribute to the scientific community a set of findings that we hope will prove helpful in enabling predictive analytics in both existing DCDTs and future projects. Starting from a thorough investigation into the new, emerging field of datacenter digital twinning, we designed a system capable of incorporating sophisticated data analysis techniques. We ended our project with a novel evaluation method used in a set of exhaustive experiments. We answer the main research question by addressing each sub-research question.

6.1 Answers to Each Research Question

RQ₁ *How to asses the current state-of-the-art of digital twinning for datacenters?*
To answer this research question, we conducted a semi-structured literature review. Our findings indicate that the field of datacenter digital twinning is still under development, and there exist few DCDT deployments. The current efforts in modelling datacenters focus on very specialized parts of datacenter management, *i.e.,* cooling and heat modelling, network mapping. Many crucial features, inherent to the Digital Twin (DT) definition are still missing from current DCDTs. Standalone DCDT systems fail to offer the holistic capabilities envisioned by the inventors of DTs. The results of the literature survey are in Table 2.2, which contains systems we found through a semi-structured literature review process. We first used structured queries, followed by a mix of snowballing and manual search. As a result of the findings from Table 2.2, we additionally provide a holistic system model that encompasses the features of all the systems from Table 2.2 (see Figure 2.4). This system model organizes the scattered, standalone systems into a single diagram. We hope it will prove useful to future researchers navigating the field of datacenter digital twinning.

RQ₂ *How to design a reference architecture for a predictive datacenter digital twin using discrete-event simulation?*

To answer this research question, we first brainstormed the potential use-cases for a predictive DCDT. The use-cases (see Section 3.2.2) are based on the findings of our literature survey. Based on them, we created a set of functional and non-functional requirements (see Sections 3.2.3 and 3.2.4) to guide our system design. Using the *AtLarge Design Process* [17] we created the reference architecture that enables predictive analysis for datacenter operators through digital twinning. Our system contains 4 major elements: (1) the datacenter (physical twin), (2) the digital thread, (3) the digital twin, and (4) predictive analytics. For a detailed discussion of the design, see Section 3.3.

RQ₃ *How to validate and evaluate a datacenter digital twin architecture in relation to system requirements?*

To answer the last research question we crated a prototype. During the prototype design, we used state-of-the-practice software, such as `Confluent Kafka`, `Redis` and `PostgreSQL` (see Section 4.2). However, as it turns out, evaluating DCDTs is not a trivial task. Lacking the physical datacenter to experiment with, we came up with a novel digital twin evaluation method. Our method, relies solely on discrete-event simulation to model the physical datacenter, overcoming the problems of real-world experimentation (*e.g.,* sustainability, costliness, reproducibility). The findings indicate that *Sunfish* (SF) can reliably differentiate between large host failures and insignificant downtime using predictions from `OpenDC`, a state-of-the-art datacenter modelling software. Moreover, we show that SF can be used to incorporate predictive analytics systems and significantly lower the total number of task failures during a workload.

6.2 Future Work

The hardware required to run future Artificial Intelligence (AI) models will more heterogeneous and power-hungry than ever before [3]. Due to the end of Dennard's scaling and the fading of Moore's law, we expect datacenter compute to incorporate more sophisticated architectures, (*e.g.,* GPUs, NPUs, TPUs) (see Figure 6.1). To tackle datacenter diversification, DCDTs are urgently needed. We envision DCDTs as systems that encompass features necessary to model the entire datacenter. To take advantage of all benefits of DT-ing, a DCDT must become a holistic, widely-employed tool. If successful, DTs will be indispensable in datacenter management, given the current growth of AI and the diversification of compute under way. To achieve the National Academy of Sciences, Engineering, and Medicine (NASEM) goals of digital twinning [39], we suggest several directions future work should take.

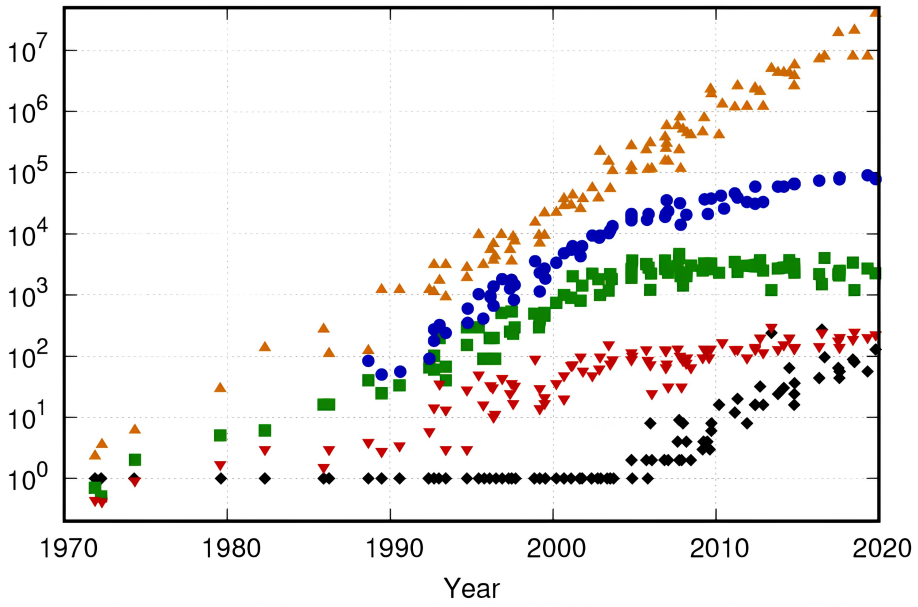


Figure 6.1: 48 years of microprocessor trend data. Legend: ▲ Transistors (thousands), ● Single Thread Performance (SpecINT $\times 10^3$), ■ Frequency (MHz), ▼ Typical Power (Watts), ◆ Number of Logical Cores [67].

6.2.1 A New, Strong Principle of DCDT Design

1. The Future Goal

Sunfish enables DCDTs to incorporate predictive analysis into facility management. Naturally, we envision upcoming DCDTs will use sophisticated prediction techniques (*e.g.,* statistical methods, Machine Learning (ML)). A DCDT must posses predictive capabilities, by definition [39].

2. What Is Missing?

To power the predictions, we envision an ML-based inference engine as a necessary component of digital twinning. The need for ML arises naturally in scenarios where large volumes of data, requiring little to no preprocessing meet the demand for estimating future facility behaviour [40, 68]. However, currently there are no DCDT deployments that model the warehouse using an ML approach to predict events (see Table 2.2).

3. The Next Steps

In short, we stipulate DCDTs should include ML in their Operational Data Analysis (ODA) analysis. The next steps would involve using *Sunfish* to employ a ML-based inference engine. To achieve this we propose to use battle-tested ML algorithms, Empirical Risk Minimization (ERM), the Support Vector Machine (SVM), linear regression, Bayesian liner regression *etc.* For future work in failure prediction, we envision an Approximate Bayesian Computation (ABC) approach to estimate the

real failure distribution within the datacenter. The clear benefits of this approach stem from the availability and ease of access to large volumes of data. Moreover, ML models are much quicker to train than Computational Fluid Dynamics (CFD)-based models, and have already been employed for other purposes in existing DCDTs [46]. There are few drawbacks to using ML in this scenario. Existing works, envision going a step further, to even employ AI-based inference engines [3]. Presently, no other statistical method can approach the accuracy of a good ML model.

6.2.2 A Crucial Step to Long-Term Success

1. The Future Goal

In Section 6.2 we highlighted the paradigm shift in datacenter design. Computer Systems is a fast-paced, dynamic field, and we need educated engineers to make correct, well-informed decisions using DCDTs. We stipulate DCDTs, specifically using simulation-based experimentation, should be included in higher education and academia (alike discrete-event simulation [18]). New, young students should be made aware of the different Computer Systems trade-offs and encouraged to experiment with DCDT tools on their own [4].

2. What Is Missing?

Currently, *Sunfish* is not an education-ready program. While a pleasant User Interface (UI) exists in the form of a `Grafana` dashboard, *Sunfish* does not facilitate an intuitive UI to experiment and explore with the DCDT capabilities. A clear, graphical UI, since Douglas Engelbrat first demonstrated it in 1968 [69], is an indispensable part of many computer programs. Presently, it is missing from *Sunfish*, and from many other DT deployments [47, 46, 15, 12] (excluding the visualization dashboards).

3. The Next Steps

In summary, we propose DCDTs be enhanced with a friendly, education-supportive UI. The next steps to achieve this are DT-specific. For example, to include a UI with *Sunfish* we envision exploring the existing Kotlin UI libraries (*e.g.,* Jetpack Compose [70]). We suggested creating either a Graphical User Interface (GUI) or a Terminal User Interface (TUI) for high accessibility. The benefits of sharing DCDTs to higher-education and academia are vast, and there are few to none drawbacks. Master-level courses on DT design already exist (see https://studiegids.vu.nl/en/courses/2026-2027/XMU_0068). We believe including DCDT experimentation in university courses can only prove beneficial to the Computer Science society.

6.2.3 Real-World Survey of Datacenter Digital Twinning

1. The Future Goal

In Chapter 2 we surveyed the literature on datacenter digital twinning. We followed the steps outlined by Kitchenham *et al.* [16], however we did not conduct any real-world interviews. Real-world practice can be much different from state-of-the-art

published in scientific-journals [71]. We propose a series of interviews with industry practitioners to gain a fuller insight into the potential use-cases of DCDTs. We stipulate a systematic literature survey, alongside qualitative interviews can offer much benefit to the DCDT community.

2. What Is Missing?

In this thesis we have conducted a simple, comprehensive literature survey of DCDTs. There already exist systematic literature surveys of generic DTs [8], however what is still missing is a systematic literature survey supported by real-world interviews and careful analysis of the DCDT subdomain. Microsoft already offers digital twinning as a service (see https://azure.microsoft.com/en-us/products/digital-twins/), and NVIDIA is deploying digital twins as well (see https://www.nvidia.com/en-sg/omniverse/). However, there exists little knowledge on state-of-the-practice (*e.g.,* datacenters using the 6SigmaDC software, NVIDIA Omniverse, Microsoft Azure Digital Twins). To develop useful digital twins, engineers must be aware of the practical challenges and any tacit knowledge associated with DCDTs.

3. The Next Steps

In short, there is no systematic literature survey of DCDTs completed alongside real-world interviews. The next steps to start the literature survey is to plan, conduct and analyze interviews with practitioners from a wide variety of background and nationalities. For an excellent example of such survey, in the field of datacenter capacity planning, we advise to read "Capelin: Fast Data-Driven Capacity Planning for Cloud Datacenters" by Andreadis *et al.* [71]. We believe the scientific research community can greatly benefit from a more holistic view of DCDTs. Most importantly, the survey could provide valuable insight into practical DCDT deployments using proprietary tools (*e.g.,* Microsoft Azure Digital Twins).

Acronyms

ABC Approximate Bayesian Computation. 51

AI Artificial Intelligence. 1, 6, 16, 17, 23, 27, 50, 52

CERN Conseil Européen pour la Recherche Nucléaire. 9, 10

CFD Computational Fluid Dynamics. 17, 52

CPU Central Processing Unit. 9, 35, 36, 48

DBMS Database Management System. 29

DCDT Datacenter Digital Twin. 3–7, 11, 14, 15, 17–19, 21–24, 28, 29, 39, 41, 42, 47, 49–53

DT Digital Twin. 2–6, 12, 13, 15, 18, 22, 28, 30, 31, 33, 36, 37, 49, 50, 52, 53

ERM Empirical Risk Minimization. 51

FAIR Findability, Accessibility, Interoperability and Reusability. 7

GUI Graphical User Interface. 52

HP Hewlett Packard. 17

HPC High Performance Computing. 13

HTTP Hyper-text Transfer Protocol. 29, 34, 36

IoT Internet-of-Things. 3

IT Information Technology. 10, 11, 19

JSON JavaScript Object Notation. 17

LLMS Large Language Models. 17

MAE Mean Absolute Error. 17

ML Machine Learning. 13, 16, 17, 27, 51, 52

NASA National Aeronautics and Space Agency. 3

NASEM National Academy of Sciences, Engineering, and Medicine. 14, 18, 19, 22, 23, 50

NPU Neural Processing Unit. 9

ODA Operational Data Analysis. 4, 6, 7, 13–15, 22, 51

POD Proper Orthogonal Decomposition. 17

PUE Power Usage Effectiveness. 26

RAM Random Access Memory. 9, 35

RL Reinforcement Learning. 16

SF *Sunfish*. 22, 30, 33, 37, 50

SLA Service Level Agreements. 3, 4, 11, 20–22, 28, 40, 41

SVM Support Vector Machine. 51

TPU Tensor Processing Unit. 9

TUI Terminal User Interface. 52

UI User Interface. 32, 52

Appendices

Appendix A

Reproducibility

Artifact Checklist

- ✓ Program(s): `OpenDC`, `Confluent Kafka`, `PostgreSQL`, `Redis`, `Grafana`.
- ✓ Data set(s): Failure traces from Talluri *et al.* and failure models from Javadi *et al.* (see also https://opendc.org/learn/documentation/Input/FailureModel).
- ✓ Experiments: in Chapter 5.
- ✓ Time is needed to reproduce the experiments: around 4 hours.
- ✓ Public Experiment Archive: https://github.com/M-J-Kwiatkowski/opendc and https://github.com/M-J-Kwiatkowski/sunfish.

Experimental Setup

Confluent Kafka, (see https://www.confluent.io/) must be present on the system. This includes the two different connectors from Chapter 4. The scripts to start the Confluent Kafka (assuming installation in `/opt/confluent` instance are listed below. First, start Kafka Connect:

```
/opt/confluent/bin/connect-standalone \
/opt/confluent/etc/kafka/connect-standalone.properties \
/opt/confluent/share/confluent-common/connectors/sink-jdbc.properties \
/opt/confluent/share/confluent-common/connectors/sink-redis.properties
```

Then, format Kafka Storage:

```
/opt/confluent/bin/kafka-storage format -t \
2vi2WtHxQAOPyXb1Bj1Jvw -c /opt/confluent/etc/kafka/server.properties \
--standalone > /dev/null 2>&1; echo 0
```

Afterwards, start Kafka Broker:

```
/opt/confluent/bin/kafka-server-start /opt/confluent/etc/kafka/server.properties
```

Then create a new Kafka Topic (*e.g.,* `postgres_topic`):

```
kafka-topics --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1
```

Lastly, run the Schema Registry:

```
/opt/confluent/bin/schema-registry-start \
/opt/confluent/etc/schema-registry/schema-registry.properties
```

At this point, ensure `PostgreSQL`, `Redi` and `Grafana` are up and running on the system, on their default port configurations. To see metrics flow seamlessly into `PostgreSQL`, once can run the following command (*i.e.,* assuming the database name is `opendc`):

```
sudo su postgres; psql -d opendc
```

Additionally, to interact with the `Redis` cache, we suggest to use the excellent `redis-cli` tool:

```
redis-cli -h localhost -p 6379
```

The following commands can be used to list the contents of the cache (*i.e.,* assuming stream name `postgres_topic`), and to clear the cache respectively:

```
XRANGE postgres_topic - +
XTRIM postgres_topic MAXLEN 0
```

For the predictive analytics in our implementation it is useful to create a separate consumer group:

```
XGROUP CREATE mystream mygroup 0
```

Lastly, each time you change the database schema, you must run (*i.e.,* assuming `schema.proto` lives in `resources/experiments`):

```
cd resources/experiments/;
protoc --java_out=/home/matt/git/opendc/opendc-common/src/main/java/  schema.proto
```

Lastly, one has to checkout the https://git.denounce.ai/opendc.git and https://git.denounce.ai/sunfish.git repositories on their local machine. `sunfish.git` acts as the digital twin and `opendc.git` acts as the physical datacenter. Both contain modifications. Change the directory to `sunfish.git`. Create a `.venv` in `python_scripts` and `http_server` and install all the dependencies. Then, run respectively the HTTP Server and the Python Scripts. Afterwards, start *Sunfish* using IntellijIDEA.

Configuration files

```
{
    "name": "greenifier-demo-scaling",
    "topologies": [
        {
            "pathToFile": "resources/topologies/surf.json"
        }
    ],
    "workloads": [
        {
            "pathToFile": "resources/workloads/surf_month",
            "type": "ComputeWorkload"
        }
    ],
    "failureModels" : [
        {
            "type": "trace-based",
            "pathToFile" : "resources/failures/Whatsapp_user_reported.parquet"
        }
    ],
    "exportModels": [
        {
            "exportInterval": 3600
        }
    ]
}
```

Listing A.1: The default experiment configuration. Here the trace is set to WhatsApp's.

```
{"clusters": [
        {
            "name": "C01",
            "hosts" : [
                {
                    "name": "H01",
                    "cpu": {
                        "coreCount": 16,
                        "coreSpeed": 2100
                    },
                    "memory": {
                        "memorySize": 128000000
```

```
                    },
                    "count": 277
                }],
            "powerSource": {
                "carbonTracePath": "resources/carbon_traces/NL_2021-2024.
    parquet"
            }
}]}
```

Listing A.2: The default toplogy used in all experiments.

Bibliography

[1] Alexandru Iosup, Fernando Kuipers, Ana Lucia Varbanescu, Paola Grosso, Animesh Trivedi, Jan S. Rellermeyer, Lin Wang, Alexandru Uta, and Francesco Regazzoni. Future computer systems and networking research in the netherlands: A manifesto. *CoRR*, abs/2206.03259, 2022. URL https://doi.org/10.48550/arXiv.2206.03259. This BibTeX citation comes from https://dblp.org/rec/journals/corr/abs-2206-03259.html?view=bibtex.

[2] Dejan S. Milojicic, Paolo Faraboschi, Nicolas Dubé, and Duncan Roweth. Future of HPC: diversifying heterogeneity. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2021, Grenoble, France, February 1-5, 2021*, pages 276–281. IEEE, 2021. URL https://doi.org/10.23919/DATE51398.2021.9474063. This BibTeX citation comes from https://dblp.org/rec/conf/date/MilojicicFDR21.html?view=bibtex.

[3] Jyotika Athavale, Cullen E. Bash, Wesley Brewer, Matthias Maiterth, Dejan S. Milojicic, Harry Petty, and Soumyendu Sarkar. Digital twins for data centers. *Computer*, 57 (10):151–158, 2024. URL https://doi.org/10.1109/MC.2024.3436945. This BibTeX citation comes from https://dblp.org/rec/journals/computer/AthavaleBBMMPS24.html?view=bibtex.

[4] Alexandru Iosup, Alexandru Uta, Laurens Versluis, Georgios Andreadis, Erwin Van Eyk, Tim Hegeman, Sacheendra Talluri, Vincent van Beek, and Lucian Toader. Massivizing computer systems: A vision to understand, design, and engineer computer ecosystems through and beyond modern distributed systems. In *38th IEEE International Conference on Distributed Computing Systems, ICDCS 2018, Vienna, Austria, July 2-6, 2018*, pages 1224–1237. IEEE Computer Society, 2018. URL https://doi.org/10.1109/ICDCS.2018.00122. This BibTeX citation comes from my 1st supervisor.

[5] Sacheendra Talluri, Leon Overweel, Laurens Versluis, Animesh Trivedi, and Alexandru Iosup. Empirical characterization of user reports about cloud failures. In Esam El-Araby, Vana Kalogeraki, Danilo Pianini, Fr édéric Lassabe, Barry Porter, Sona Ghahremani, Ingrid Nunes, Mohamed Bakhouya, and Sven Tomforde, editors, *IEEE International Conference on Autonomic Computing and Self-Organizing Systems, ACSOS 2021, Washington, DC, USA, September 27 - Oct. 1, 2021*, pages 158–163.

IEEE, 2021. URL https://doi.org/10.1109/ACSOS52086.2021.00039. This BibTeX citation comes from https://dblp.org/rec/conf/acsos/TalluriOVTI21.html?view=bibtex.

[6] Douglas Donnellan, Andy Lawrence, and Rose Weinshenk. Executive summary: Annual outage analysis 2025, May 2025. URL https://uptimeinstitute.com/resources/research-and-reports/annual-outage-analysis-2025. This citation has been written by me. The Uptime Institue (see uptimeinstitute.com) is a renowned institution that certifies datacenters. For more information we advise to look at https://en.wikipedia.org/wiki/451_Group#Uptime_Institute,.

[7] Wikipedia contributors. Digital twin — Wikipedia, the free encyclopedia, 2026. URL https://en.wikipedia.org/w/index.php?title=Digital_twin&oldid=1351132391. This BibTeX citation comes from https://en.wikipedia.org/w/index.php?title=Special:CiteThisPage&page=Digital_twin&id=1359969562&wpFormIdentifier=titleform#BibTeX_entry.

[8] Fei Tao, Meng Zhang, Yushan Liu, and A.Y.C. Nee. Digital twin driven prognostics and health management for complex equipment. *CIRP Annals*, 67(1): 169–172, 2018. ISSN 0007-8506. URL https://www.sciencedirect.com/science/article/pii/S0007850618300799. This BibTeX citation comes from https://www.sciencedirect.com/science/article/abs/pii/S0007850618300799?via%3Dihub.

[9] Eric J. Tuegel, Anthony R. Ingraffea, Thomas G. Eason, and S. Michael Spottswood. Reengineering aircraft structural life prediction using a digital twin. *International Journal of Aerospace Engineering*, 2011(1):154798, 2011. URL https://onlinelibrary.wiley.com/doi/abs/10.1155/2011/154798. This BibTeX citation comes from https://onlinelibrary.wiley.com/doi/full/10.1155/2011/154798.

[10] Eric Tuegel. The airframe digital twin: Some challenges to realization. 04 2012. URL https://doi.org/10.2514/6.2012-1812. This citation has been created by me. See https://arc.aiaa.org/doi/abs/10.2514/6.2012-1812 for more details about the conference and the article. AIAA is a world-class aerospace engineering association.

[11] Magdalena Görtz. Digital twins: past, present and future. *Scientific Reports*, 16(1):10510, Mar 27, 2026. ISSN 2045-2322. URL https://doi.org/10.1038/s41598-026-45272-z. This RIS citation comes from https://www.nature.com/articles/s41598-026-45272-z#citeas.

[12] Wesley Brewer, Matthias Maiterth, Vineet Kumar, Rafal P. Wojda, Sedrick Bouknight, Jesse Hines, Woong Shin, Scott Greenwood, David Grant, Wesley Williams, and Feiyi Wang. A digital twin framework for liquid-cooled supercomputers as demonstrated at exascale. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, SC 2024, Atlanta, GA, USA, November 17-22, 2024*, page 23. IEEE, 2024. URL

https://dl.acm.org/doi/10.1109/SC41406.2024.00029. This BibTeX citation comes from https://dblp.org/rec/conf/sc/BrewerMKWBHSGGW24.html?view=bibtex.

[13] Shekhar Suman, Xiaoyu Chu, Dante Niewenhuis, Sacheendra Talluri, Tiziano De Matteis, and Alexandru Iosup. Enabling operational data analytics for datacenters through ontologies, monitoring, and simulation-based prediction. In Simonetta Balsamo, William J. Knottenbelt, Cristina L. Abad, and Weiyi Shang, editors, *Companion of the 15th ACM/SPEC International Conference on Performance Engineering, ICPE 2024, London, United Kingdom, May 7-11, 2024*, pages 120–126. ACM, 2024. URL https://doi.org/10.1145/3629527.3652897. This BibTeX citation comes from https://dblp.org/rec/conf/wosp/SumanCNTMI24.html?view=bibtex.

[14] Alexandru Iosup. A vu on digital twins to improve the performance and technological sustainability of datacenters in the continuum. MODSIM World, Seattle, US, 2024. URL https://atlarge-research.com/talks/2023-modsim-alex.html. This citation has been created by me. See the URL for more details about the talk.

[15] Ebad Taheri, Pedro Bruel, Pavana Prakash, Gourav Rattihalli, Ninad Hogade, Aditya Dhakal, Rolando P. Hong Enriquez, Torsten Wilde, Leo Popokh, Dejan S. Milojicic, and Cullen E. Bash. Dytwin: Federated adaptive digital twins for data centers - visualization and anomaly detection. In *SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis , Atlanta, GA, USA, November 17-22, 2024*, pages 847–852. IEEE, 2024. URL https://doi.org/10.1109/SCW63240.2024.00120. This BibTeX citation comes from https://dblp.org/rec/conf/sc/TaheriBPRHDEWPM24.html?view=bibtex.

[16] Barbara A. Kitchenham, Rialette Pretorius, David Budgen, Pearl Brereton, Mark Turner, Mahmood Niazi, and Stephen G. Linkman. Systematic literature reviews in software engineering - A tertiary study. *Inf. Softw. Technol.*, 52(8):792–805, 2010. URL https://doi.org/10.1016/j.infsof.2010.03.006. This BibTeX citation comes from https://dblp.org/rec/journals/infsof/KitchenhamPBBTNL10.html?view=bibtex.

[17] Alexandru Iosup, Laurens Versluis, Animesh Trivedi, Erwin Van Eyk, Lucian Toader, Vincent van Beek, Giulia Frascaria, Ahmed Musaafir, and Sacheendra Talluri. The atlarge vision on the design of distributed systems and ecosystems. In *39th IEEE International Conference on Distributed Computing Systems, ICDCS 2019, Dallas, TX, USA, July 7-10, 2019*, pages 1765–1776. IEEE, 2019. URL https://doi.org/10.1109/ICDCS.2019.00175. This BibTeX citation comes from https://dblp.org/rec/conf/icdcs/IosupVTETBFMT19.html?view=bibtex .

[18] Fabian Mastenbroek, Georgios Andreadis, Soufiane Jounaid, Wenchen Lai, Jacob Burley, Jaro Bosch, Erwin Van Eyk, Laurens Versluis, Vincent van Beek, and Alexandru Iosup. Opendc 2.0: Convenient modeling and simulation of emerging technologies in cloud datacenters. In Laurent Lefèvre, Stacy Patterson, Young Choon

Lee, Haiying Shen, Shashikant Ilager, Mohammad Goudarzi, Adel Nadjaran Toosi, and Rajkumar Buyya, editors, *21st IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing, CCGrid 2021, Melbourne, Australia, May 10-13, 2021*, pages 455–464. IEEE, 2021. URL https://doi.org/10.1109/CCGrid51090.2021.00055. This BibTeX citation comes from https://dblp.org/rec/conf/ccgrid/MastenbroekAJLB21.html?view=bibtex.

[19] Wikipedia contributors. Data center — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Data_center&oldid=1364097634, 2026. This BibTeX citation comes from https://en.wikipedia.org/w/index.php?title=Special:CiteThisPage&page=Data_center&id=1364097634&wpFormIdentifier=titleform#BibTeX_entry.

[20] Wikipedia contributors. Jevons paradox — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Jevons_paradox&oldid=1363548499, 2026. This BibTeX citation comes from https://en.wikipedia.org/w/index.php?title=Special:CiteThisPage&page=Jevons_paradox&id=1363548499&wpFormIdentifier=titleform#BibTeX_entry.

[21] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice: the condor experience. *Concurrency and Computation: Practice and Experience*, 17(2-4):323–356, 2005. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.938. This BibTeX citation comes from https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.938.

[22] Wikipedia contributors. Systems thinking — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Systems_thinking&oldid=1360310082, 2026. This BibTeX citation comes from: https://en.wikipedia.org/w/index.php?title=Special:CiteThisPage&page=Systems_thinking&id=1360310082&wpFormIdentifier=titleform#BibTeX_entry.

[23] Bahman Javadi, Derrick Kondo, Alexandru Iosup, and Dick H. J. Epema. The failure trace archive: Enabling the comparison of failure measurements and models of distributed systems. *J. Parallel Distributed Comput.*, 73(8):1208–1223, 2013. URL https://doi.org/10.1016/j.jpdc.2013.04.002. This BibTeX citation comes from https://dblp.org/rec/journals/jpdc/JavadiKIE13.html?view=bibtex.

[24] Wikipedia contributors. Downtime — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Downtime&oldid=1360763149, 2026. This BibTeX citation comes from https://en.wikipedia.org/w/index.php?title=Special:CiteThisPage&page=Downtime&id=1360763149&wpFormIdentifier=titleform#BibTeX_entry.

[25] Fabian Mastenbroek, Georgios Andreadis, Soufiane Jounaid, Wenchen Lai, Jacob Burley, Jaro Bosch, Erwin van Eyk, Laurens Versluis, Vincent van Beek, and

Alexandru Iosup. Opendc. This BibTeX citation comes from https://github.com/atlarge-research/opendc. It has been converted from a CITATION.cff file using cffconvert (see https://citation-file-format.github.io/).

[26] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, César A. F. De Rose, and Rajkumar Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw. Pract. Exp.*, 41(1):23–50, 2011. URL https://doi.org/10.1002/spe.995. This BibTeX citation comes from https://dblp.org/rec/journals/spe/CalheirosRBRB11.html?view=bibtex.

[27] Harshit Gupta, Amir Vahid Dastjerdi, Soumya K. Ghosh, and Rajkumar Buyya. ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments. *Softw. Pract. Exp.*, 47(9): 1275–1296, 2017. URL https://doi.org/10.1002/spe.2509. This BibTeX citation comes from https://dblp.org/rec/journals/spe/GuptaDGB17.html?view=bibtex.

[28] Weiwei Chen and Ewa Deelman. Workflowsim: A toolkit for simulating scientific workflows in distributed environments. In *8th IEEE International Conference on E-Science, e-Science 2012, Chicago, IL, USA, October 8-12, 2012*, pages 1–8. IEEE Computer Society, 2012. URL https://doi.org/10.1109/eScience.2012.6404430. This BibTeX citation comes from https://dblp.org/rec/conf/eScience/ChenD12.html?view=bibtex.

[29] Bhathiya Wickremasinghe, Rodrigo N. Calheiros, and Rajkumar Buyya. Cloudanalyst: A cloudsim-based visual modeller for analysing cloud computing environments and applications. In *24th IEEE International Conference on Advanced Information Networking and Applications, AINA 2010, Perth, Australia, 20-13 April 2010*, pages 446–452. IEEE Computer Society, 2010. URL https://doi.org/10.1109/AINA.2010.32. This BibTeX citation comes from https://dblp.org/rec/conf/aina/WickremasingheCB10.html?view=bibtex.

[30] Henri Casanova, Arnaud Giersch, Arnaud Legrand, Martin Quinson, and Frédéric Suter. Simgrid: a sustained effort for the versatile simulation of large scale distributed systems. *CoRR*, abs/1309.1630, 2013. URL http://arxiv.org/abs/1309.1630. This BibTeX citation comes from https://dblp.org/rec/journals/corr/CasanovaGLQS13.html?view=bibtex.

[31] Etienne Michon, Julien Gossa, Stéphane Genaud, Léo Unbekandt, and Vincent Kherbache. Schlouder: A broker for iaas clouds. *Future Gener. Comput. Syst.*, 69:11–23, 2017. URL https://doi.org/10.1016/j.future.2016.09.010. This BibTeX citation comes from https://dblp.org/rec/journals/fgcs/MichonGGUK17.html?view=bibtex.

[32] Henri Casanova, Ryan Tanaka, William Koch, and Rafael Ferreira da Silva. Teaching parallel and distributed computing concepts in simulation with WRENCH. *J. Parallel Distributed Comput.*, 156:53–63, 2021. URL https://doi.org/10.1016/j.jpdc.2021.05.009. This BibTeX citation comes from https://dblp.org/rec/journals/jpdc/CasanovaTKS21.html?view=bibtex.

[33] Takahiro Hirofuchi, Adrien Lebre, and Laurent Pouilloux. Simgrid VM: virtual machine support for a simulation framework of distributed systems. *IEEE Trans. Cloud Comput.*, 6(1):221–234, 2018. URL https://doi.org/10.1109/TCC.2015.2481422. This BibTeX citation comes from https://dblp.org/rec/journals/tcc/HirofuchiLP18.html?view=bibtex.

[34] Henri Casanova, Rafael Ferreira da Silva, Ryan Tanaka, Suraj Pandey, Gautam Jethwani, William Koch, Spencer Albrecht, James Oeth, and Frédéric Suter. Developing accurate and scalable simulators of production workflow management systems with WRENCH (version 2.0). https://doi.org/10.5281/zenodo.7158509, November 2020. URL https://doi.org/10.5281/zenodo.7158509. Accessed on YYYY-MM-DD.

[35] Alexandru Iosup, Omer Ozan Sonmez, and Dick H. J. Epema. Dgsim: Comparing grid resource management architectures through trace-based simulation. In Emilio Luque, Tomàs Margalef, and Domingo Benitez, editors, *Euro-Par 2008 - Parallel Processing, 14th International Euro-Par Conference, Las Palmas de Gran Canaria, Spain, August 26-29, 2008, Proceedings*, Lecture Notes in Computer Science, pages 13–25. Springer, 2008. URL https://doi.org/10.1007/978-3-540-85451-7_3. This BibTeX citation comes from https://dblp.org/rec/conf/europar/IosupSE08.html?view=bibtex.

[36] Simon Ostermann, Kassian Plankensteiner, Radu Prodan, and Thomas Fahringer. Groudsim: An event-based simulation framework for computational grids and clouds. In Mario R. Guarracino, Frédéric Vivien, Jesper Larsson Träff, Mario Cannataro, Marco Danelutto, Anders Hast, Francesca Perla, Andreas Knüpfer, Beniamino Di Martino, and Michael Alexander, editors, *Euro-Par 2010 Parallel Processing Workshops - HeteroPar, HPCC, HiBB, CoreGrid, UCHPC, HPCF, PROPER, CCPI, VHPC, Ischia, Italy, August 31-September 3, 2010, Revised Selected Papers*, Lecture Notes in Computer Science, pages 305–313. Springer, 2010. URL https://doi.org/10.1007/978-3-642-21878-1_38. This BibTeX citation comes from https://dblp.org/rec/conf/europar/OstermannPPF10.html?view=bibtex.

[37] Alberto Nuñez, José Luis Vázquez- Poletti, Agustín C. Caminero, Gabriel G. Castañé, Jesús Carretero, and Ignacio Martín Llorente. icancloud: A flexible and scalable cloud infrastructure simulator. *J. Grid Comput.*, 10(1):185–209, 2012. URL https://doi.org/10.1007/s10723-012-9208-5. This BibTeX citation comes from https://dblp.org/rec/journals/grid/NunezVCCCL12.html?view=bibtex.

[38] Jerry Banks, John S. Carson II, Barry L. Nelson, and David M. Nicol. *Discrete-Event System Simulation, 5th New Internatinal Edition*. Pearson Education, 2010. ISBN 978-1-292-02437-0. This BibTeX citation comes from https://dblp.org/rec/books/daglib/0034857.html?view=bibtex.

[39] National Academy of Engineering, National Academies of Sciences Engineering, and Medicine. Foundational research gaps and future directions for digital twins. The National Academies Press, Washington, DC, 2024. ISBN 978-0-309-70042-9. URL https://nap.nationalacademies.org/catalog/26894/foundational-research-gaps-and-future-directions-for-digital-twins. This citation has been created by me. See the ISBN for more details about the publication.

[40] Wikipedia contributors. Predictive modelling — Wikipedia, the free encyclopedia, 2026. URL https://en.wikipedia.org/w/index.php?title=Predictive_modelling\&oldid=1334478640. This BibTeX citations comes from https://en.wikipedia.org/w/index.php?title=Special:CiteThisPage&page=Predictive_modelling&id=1362368215&wpFormIdentifier=titleform#BibTeX_entry.

[41] Norman Bourassa, Walker Johnson, Jeff Broughton, Deirdre McShane Carter, Sadie Joy, Raphael Vitti, and Peter Seto. Operational data analytics: Optimizing the national energy research scientific computing center cooling systems. In *48th International Conference on Parallel Processing, ICPP 2019 Workshop Proceedings, Kyoto, Japan, August 05-08, 2019*, pages 5:1–5:7. ACM, 2019. URL https://doi.org/10.1145/3339186.3339210. This BibTeX citation comes from https://dblp.org/rec/conf/icppw/BourassaJBCJVS19.html?view=bibtex.

[42] Alessio Netti, Micha Müller, Carla Guillén, Michael Ott, Daniele Tafani, Gence Ozer, and Martin Schulz. DCDB wintermute: Enabling online and holistic operational data analytics on HPC systems. In Manish Parashar, Vladimir Vlassov, David E. Irwin, and Kathryn M. Mohror, editors, *HPDC '20: The 29th International Symposium on High-Performance Parallel and Distributed Computing, Stockholm, Sweden, June 23-26, 2020*, pages 101–112. ACM, 2020. URL https://doi.org/10.1145/3369583.3392674. This BibTeX citation comes from https://dblp.org/rec/conf/hpdc/NettiMGOTO020.html?view=bibtex.

[43] Andrea Borghesi, Martin Molan, Michela Milano, and Andrea Bartolini. Anomaly detection and anticipation in high performance computing systems. *IEEE Trans. Parallel Distributed Syst.*, 33(4):739–750, 2022. URL https://doi.org/10.1109/TPDS.2021.3082802. This BibTeX citation comes from https://dblp.org/rec/journals/tpds/BorghesiMMB22.html?view=bibtex.

[44] Umit Demirbaga, Zhenyu Wen, Ayman Noor, Karan Mitra, Khaled Alwasel, Saurabh Garg, Albert Y. Zomaya, and Rajiv Ranjan. Autodiagn: An automated real-time diagnosis framework for big data systems. *IEEE Trans. Computers*, 71(5):1035–1048,

2022. URL https://doi.org/10.1109/TC.2021.3070639. This BibTeX citation comes from https://dblp.org/rec/journals/tc/DemirbagaWNMAGZ22.html?view=bibtex.

[45] Jane Webster and Richard T. Watson. Analyzing the past to prepare for the future: writing a literature review. *MIS Q.*, 26(2):xiii–xxiii, June 2002. ISSN 0276-7783. This BibTeX citation comes from https://dl.acm.org/doi/10.5555/2017160.2017162.

[46] Ziting Zhang, Yu Zeng, Haoran Liu, Chaoyue Zhao, Feng Wang, and Yunqing Chen. Smart DC: an AI and digital twin-based energy-saving solution for data centers. In *2022 IEEE/IFIP Network Operations and Management Symposium, NOMS 2022, Budapest, Hungary, April 25-29, 2022*, pages 1–6. IEEE, 2022. URL https://doi.org/10.1109/NOMS54207.2022.9789853. This BibTeX citation comes from https://dblp.org/rec/conf/noms/ZhangZLZWC22.html?view=bibtex.

[47] Minghao Li, Ruihang Wang, Xin Zhou, Zhaomeng Zhu, Yonggang Wen, and Rui Tan. Chattwin: Toward automated digital twin generation for data center via large language models. In *Proceedings of the 10th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation, BuildSys 2023, Istanbul, Turkey, November 15-16, 2023*, pages 208–211. ACM, 2023. URL https://doi.org/10.1145/3600100.3623719. This BibTeX citation comes from https://dblp.org/rec/conf/sensys/LiW0Z0T23.html?view=bibtex.

[48] Zhiwei Cao, Ruihang Wang, Xin Zhou, and Yonggang Wen. Reducio: model reduction for data center predictive digital twins via physics-guided machine learning. In Jorge Ortiz, editor, *Proceedings of the 9th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation, BuildSys 2022, Boston, Massachusetts, November 9-10, 2022*, pages 1–10. ACM, 2022. URL https://doi.org/10.1145/3563357.3564050. This BibTeX citation comes from https://dblp.org/rec/conf/sensys/CaoW0022.html?view=bibtex.

[49] Hanshu Hong, Qin Wu, Feng Dong, Wei Song, Ronghua Sun, Tao Han, Cheng Zhou, and Hongwei Yang. Netgraph: An intelligent operated digital twin platform for data center networks. In *NAI'21: Proceedings of the ACM SIGCOMM 2021 Workshop on Network-Application Integration, Virtual Event, USA, August 27, 2021*, pages 26–32. ACM, 2021. URL https://doi.org/10.1145/3472727.3472802. This BibTeX citation comes from https://dblp.org/rec/conf/sigcomm/HongWDSSHZY21.html?view=bibtex .

[50] Ruihang Wang, Xin Zhou, Linsen Dong, Yonggang Wen, Rui Tan, Li Chen, Guan Wang, and Feng Zeng. Kalibre: Knowledge-based neural surrogate model calibration for data center digital twins. In *BuildSys '20: The 7th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation, Virtual Event, Japan, November 18-20, 2020*, pages 200–209. ACM, 2020. URL https://doi.org/10.1145/3408308.3427982. This BibTeX citation comes from https://dblp.org/rec/conf/sensys/WangZD0TCWZ20.html?view=bibtex.

[51] Haitao Zhu and Botao Lin. Digital twin-driven energy consumption management of integrated heat pipe cooling system for a data center. volume 373, page 123840, 2024. ISSN 0306-2619. URL https://www.sciencedirect.com/science/article/pii/S0306261924012236. This BibTeX citation comes from https://www.sciencedirect.com/science/article/abs/pii/S0306261924012236.

[52] Wikipedia contributors. Grafana — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Grafana&oldid=1362050182, 2026. This BibTeX citation comes from: https://en.wikipedia.org/w/index.php?title=Special:CiteThisPage&page=Grafana&id=1362050182&wpFormIdentifier=titleform#BibTeX_entry.

[53] Wikipedia contributors. Confluent — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Confluent&oldid=1361669280, 2026. This BibTeX citation comes from: https://en.wikipedia.org/w/index.php?title=Special:CiteThisPage&page=Confluent&id=1361669280&wpFormIdentifier=titleform#BibTeX_entry.

[54] Wikipedia contributors. Redis — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Redis&oldid=1351092330, 2026. This BibTeX citation comes from https://en.wikipedia.org/w/index.php?title=Special:CiteThisPage&page=Redis&id=1351092330&wpFormIdentifier=titleform#BibTeX_entry.

[55] Wikipedia contributors. Postgresql — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=PostgreSQL&oldid=1357817665, 2026. This BibTeX citation comes from https://en.wikipedia.org/w/index.php?title=Special:CiteThisPage&page=PostgreSQL&id=1357817665&wpFormIdentifier=titleform#BibTeX_entry.

[56] Radu Nicolae, Jules van der Toorn, Stavriana Kraniti, Houcen Liu, and Alexandru Iosup. Opendt: Exploring datacenter performance and sustainability with a self-calibrating digital twin. In Roberto Verdecchia, Catia Trubiani, Radu Calinescu, and Ana Lucia Verbanescu, editors, *Companion of the 17th ACM/SPEC International Conference on Performance Engineering, ICPE Companion 2026, Florence, Italy, May 4-8, 2026*, pages 142–147. ACM, 2026. URL https://doi.org/10.1145/3777911.3800634. This BibTeX citation comes from https://dblp.org/rec/conf/wosp/NicolaeTKLI26.html?view=bibtex.

[57] Wikipedia contributors. Kibana — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Kibana&oldid=1342816695, 2026. This BibTeX citation comes from https://en.wikipedia.org/w/index.php?title=Special:CiteThisPage&page=Kibana&id=1342816695&wpFormIdentifier=titleform.

[58] Wikipedia contributors. Prometheus (software) — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Prometheus_(software)&oldid=1349456026, 2026. This BibTeX citation comes from https://en.wikipedia.org/w/index.php?title=Special:CiteThisPage&page=Prometheus_%28software%29&id=1349456026&wpFormIdentifier=titleform.

[59] Wikipedia contributors. Graphite (software) — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Graphite_(software)&oldid=1357227590, 2026. This BibTeX citation comes from https://en.wikipedia.org/w/index.php?title=Special:CiteThisPage&page=Graphite_%28software%29&id=1357227590&wpFormIdentifier=titleform.

[60] Wikipedia contributors. Protocol buffers — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Protocol_Buffers&oldid=1357670495, 2026. This BibTeX citation comes from https://en.wikipedia.org/w/index.php?title=Special:CiteThisPage&page=Protocol_Buffers&id=1357670495&wpFormIdentifier=titleform#BibTeX_entry.

[61] Wikipedia contributors. Memcached — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Memcached&oldid=1361835153, 2026. This BibTeX citation comes from https://en.wikipedia.org/w/index.php?title=Special:CiteThisPage&page=Memcached&id=1361835153&wpFormIdentifier=titleform#BibTeX_entry.

[62] Wikipedia contributors. Discrete-event simulation — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Discrete-event_simulation&oldid=1362915603, 2026. This BibTeX citation comes from https://en.wikipedia.org/w/index.php?title=Special:CiteThisPage&page=Discrete-event_simulation&id=1362915603&wpFormIdentifier=titleform#BibTeX_entry.

[63] Ana Maria Muscă, Alexandru Iosup, Dante Niewenhuis, and Tiziano De Matteis. An analysis of the performance and carbon footprint of workloads in datacenters using serverless models in simulation. *Vrije Universiteit Amsterdam Thesis*, 2025. URL https://www.overleaf.com/read/mbsmfjsvnmgw#7e78a7. This BibTeX citation was created by me.

[64] Siqi Shen, Vincent van Beek, and Alexandru Iosup. Statistical characterization of business-critical workloads hosted in cloud datacenters. In *15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2015, Shenzhen, China, May 4-7, 2015*, pages 465–474. IEEE Computer Society, 2015. URL https://doi.org/10.1109/CCGrid.2015.60. This BibTeX citation comes from https://dblp.org/rec/conf/ccgrid/ShenBI15.html?view=bibtex.

[65] Sacheendra Talluri, Dante Niewenhuis, Xiaoyu Chu, Jakob Kyselica, Mehmet Çetin, Alexander Balgavy, and Alexandru Iosup. Cloud uptime archive: Open-access

availability data of web, cloud, and gaming services. *IEEE Trans. Parallel Distributed Syst.*, 37(5):1136–1152, 2026. URL https://doi.org/10.1109/TPDS.2026.3673519. This BibTeX citation comes from https://dblp.org/rec/journals/tpds/TalluriNCKCBI26.html?view=bibtex.

[66] Radu Nicolae, Dante Niewenhuis, Sacheendra Talluri, and Alexandru Iosup. M3SA: exploring datacenter performance and climate-impact with multi- and meta-model simulation and analysis. In *Proceedings of the 23rd ACM International Conference on Computing Frontiers, CF 2026, Catania, Italy, May 19-21, 2026*, pages 113–117. ACM, 2026. URL https://doi.org/10.1145/3801487.3806065. This BibTeX citation comes from https://dblp.org/rec/conf/cf/NicolaeNTI26.html?view=bibtex.

[67] M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten. 48 years of microprocessor trend data, 2019. URL https://github.com/karlrupp/microprocessor-trend-data. This citation has been created by me. See the URL for more information of Karl Rupp's diagram.

[68] Marc Peter Deisenroth, A. Aldo Faisal, and Cheng Soon Ong. *Contents*, page v–viii. Cambridge University Press, 2020. This APA citations comes from https://www.cambridge.org/highereducation/books/mathematics-for-machine-learning/5EE57FD1CFB23E6EB11E130309C7EF98.

[69] Wikipedia contributors. User interface — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=User_interface&oldid=1360287461, 2026. This BibTeX citation comes from https://en.wikipedia.org/w/index.php?title=Special:CiteThisPage&page=User_interface&id=1360287461&wpFormIdentifier=titleform#BibTeX_entry.

[70] Wikipedia contributors. Jetpack compose — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Jetpack_Compose&oldid=1360694775, 2026. This BibTeX citation comes from https://en.wikipedia.org/w/index.php?title=Special:CiteThisPage&page=Jetpack_Compose&id=1360694775&wpFormIdentifier=titleform#BibTeX_entry.

[71] Georgios Andreadis, Fabian Mastenbroek, Vincent van Beek, and Alexandru Iosup. Capelin: Data-driven capacity procurement for cloud datacenters using portfolios of scenarios - extended technical report. *CoRR*, abs/2103.02060, 2021. URL https://arxiv.org/abs/2103.02060. This BibTeX citation comes from https://dblp.org/rec/journals/corr/abs-2103-02060.html?view=bibtex.