Vrije Universiteit Amsterdam

Universiteit van Amsterdam

Master Thesis

Trace Generation and Closed-Loop Digital Twin for the Digital Continuum

Author: Man Yin Edward Chui (2850095)

1st supervisor:	Alexandru Iosup
daily supervisor:	Matthijs Jansen
2nd reader:	Tiziano De Matteis

A thesis submitted in fulfillment of the requirements for the joint UvA-VU Master of Science degree in Computer Science

August 13, 2026

Abstract

Digital services increasingly operate across endpoint, edge, and cloud resources, where resource-management decisions must respond to changing workloads, heterogeneous capacity, and runtime–energy trade-offs. Testing candidate actions directly on a running system is costly and disruptive, so digital twins can provide an alternative by evaluating possible actions through a virtual representation of the physical system. However, many existing digital twins remain limited to monitoring, replay, or human-mediated feedback and lack an automated end-to-end path from observation and simulation to decision making and safe actuation. This thesis addresses this gap by designing a trace-based closed-loop digital twin architecture for the digital continuum.

The architecture connects a physical execution environment to a simulator-backed digital entity through a closed-loop control plane. The control plane records observations from the physical system as execution traces, which support workload replay and simulator calibration. From the current resource configuration, it generates alternative configurations for the simulator to evaluate. A policy selects one candidate, while explicit interfaces keep each candidate and simulation result linked to the physical state from which it was derived. We implement the architecture at the cloud-resource level, using Kubernetes for physical execution and OpenDC for simulation. The prototype defines a trace contract for replay and calibration and calibrates one parameter in OpenDC's CPU power model using recent measurements from the current worker-node configuration. The controller enacts the selected resource configuration by changing the number of existing worker nodes available for new workload placement.

We conduct controlled experiments to assess simulator fidelity, the effect of topology-aware calibration, and the decision-support value of the closed loop. OpenDC reproduces task progress and timing closely when worker capacity

is sufficient, but underestimates tail queueing and total slowdown under constrained capacity. Its processor-power predictions remain imperfect, so we treat its output as comparative evidence instead of exact physical predictions. Calibration does not consistently improve processor-power fidelity; its effect varies across settings and metrics. Relative to the reactive pending-Pod baseline, the runtime-oriented policy maintains similar runtime while reducing processor-domain energy. The power-oriented policy achieves only a modest additional energy reduction but incurs a large runtime penalty. These findings demonstrate the feasibility and decision-support value of the proposed architecture in the evaluated environment. They also show that this value depends on the fidelity of the comparison evidence, calibration scope, and policy design.

Contents

Glossary

API Application Programming Interface

DT Digital Twin

MAPE-K Monitor, Analyze, Plan, Execute using Knowledge

RAPL Running Average Power Limit

Glossary

1

Introduction

In recent years, digital services have increasingly relied on large distributed computing infrastructures, where operational decisions must be made under changing workloads and resource constraints. These decisions include routine resource management as well as responding to unexpected faults such as node failures and network degradation that reduce available resources (1, 2). Such events may require reallocating resources, migrating workloads, or temporarily degrading service levels, while balancing performance, cost, and energy consumption. However, validating response strategies directly on production systems is often impractical, as experiments can be time-consuming and expensive to set up. This motivates decision-support approaches that allow alternative actions to be evaluated quickly and safely before changes are applied to the physical system.

A **Digital Twin (DT)** provides such decision support by linking a physical system with a virtual representation (3). Such a system enables mirroring and replaying observed execution, improving reproducibility and monitoring (4). It can also support the comparison of candidate responses through a series of "what-if" analyses without disrupting production operations (5). Beyond offline analysis, a digital twin can be *closed-loop* when insights from analysis are translated into concrete control actions that adapt the physical system (3). Under such a closed loop, observations from the physical system are used to update the digital entity, while simulation results guide policy changes back to the physical system automatically.

Digital twins are a promising technology and have been adopted in several domains, including manufacturing, construction, and healthcare (3, 6, 7). The global digital twin market size is projected to grow from USD $21 billion in 2025 up to $150 billion by 2030 at a CAGR of 47.9% (8). It is also estimated that DT-based approaches can improve operational efficiency by up to 15%, and reduce costs by 20-30% (9). Virtual experimentation can

also reduce the need for physical prototyping and testing, which may contribute to lower resource use and emissions (10).

Yet applying digital twins to the management of digital services raises distinct challenges. Unlike engineered physical assets, these services are distributed and highly dynamic, which makes it difficult to obtain a virtual counterpart that is both realistic and maintainable over time. Thus, this thesis investigates how to support closed-loop decision-making using a trace-based approach, where execution traces (runtime records of system behavior) are captured and then reused to evaluate "what-if" operational decisions safely and efficiently. The following section introduces the computing context considered in this thesis and motivates trace-based simulation as a foundation for such digital twins.

1.1 Context

Digital services are increasingly deployed across the digital continuum, a paradigm in which applications and services span cloud datacenters, edge infrastructure, and endpoint devices (e.g., smartphones) (11). This continuum is increasingly common in settings such as Internet of Things applications, interactive services, and distributed data processing, where raw data and user interactions originate at the endpoint while heavier computation and long-term storage often remain in the cloud (12). Operating across these layers introduces complexity as hardware capabilities are heterogeneous, availability can fluctuate, and workloads vary over time in both volume and resource profiles. As a result, resource management decisions become critical. Decisions such as where workloads should execute, how much capacity should be allocated, and how system parameters should be adapted can strongly affect end-to-end performance, cost, and energy consumption. For example, placing computation closer to data sources may reduce latency, while assigning resource-intensive workloads to more capable infrastructure may improve throughput or energy efficiency (12, 13). These decisions must often be revisited as workloads, infrastructure conditions, and optimization goals change.

Many of these decisions are multi-objective, most notably involving a trade-off between performance and energy consumption. Improving performance does not necessarily align with reducing energy usage. Techniques that raise performance often increase power, while power-saving mechanisms can increase execution time and latency. For instance, increasing parallelism can shorten makespan while increasing power draw, whereas configurations like Dynamic Voltage and Frequency Scaling (DVFS) and low-power modes can reduce power at the expense of longer runtime and higher queueing times (14, 15). In the digital continuum,

these trade-offs become less intuitive as policies interact with heterogeneity and dynamism. Evidence-based decision support is therefore required to compare candidate policies under realistic conditions and anticipated impacts.

To create a digital entity that can support such decision making, prior work typically follows one of several modeling approaches, including mechanism/physics-based models, multiphysics models, simulation models, and data models (16, 17). For the digital continuum, mechanism and multiphysics models are less applicable as they are generally used to represent physical entities governed by laws of physics, such as electrical and mechanical interactions. Simulator-based models use abstractions to capture system behavior under varying conditions. Data models, including machine-learning approaches, learn predictive relations from historical observations and can be powerful when training data is abundant and system behavior is stable.

Among simulator-based approaches, this thesis adopts a trace-based form of simulation, where captured execution traces from the physical system are transformed into simulator-compatible inputs to enable reproducible "what-if" exploration. The motivation behind this choice is practical. In highly heterogeneous distributed environments, purely data-driven models typically require large and representative datasets and must be retrained as system configurations evolve. Collecting such data and training models at scale is costly and challenging in the digital continuum (12, 18). In contrast, execution traces provide a compact record of concrete system behavior that can be replayed and systematically altered in a simulator, enabling what-if analysis without the repeated data collection and retraining required by predictive models. We therefore treat data-driven modeling as out of scope and instead focus on how a trace-based interface, together with simulator calibration and actuation logic, can support a closed-loop workflow.

1.2 Problem Statement

While digital twins are widely regarded as a promising approach for improving monitoring and decision support, most existing research and deployments remain at the *digital model* or *digital shadow* stages. These stages differ primarily in the direction and automation of data exchange between physical and digital entities, as illustrated in Figure 1.1. In a digital model, the physical and digital entities are not connected through automated data exchange, whereas a digital shadow typically supports only one-way data flow from the physical system to its digital counterpart (17, 19). Digital models and digital shadows can help observe and reproduce behavior, but they do not reliably complete the feedback

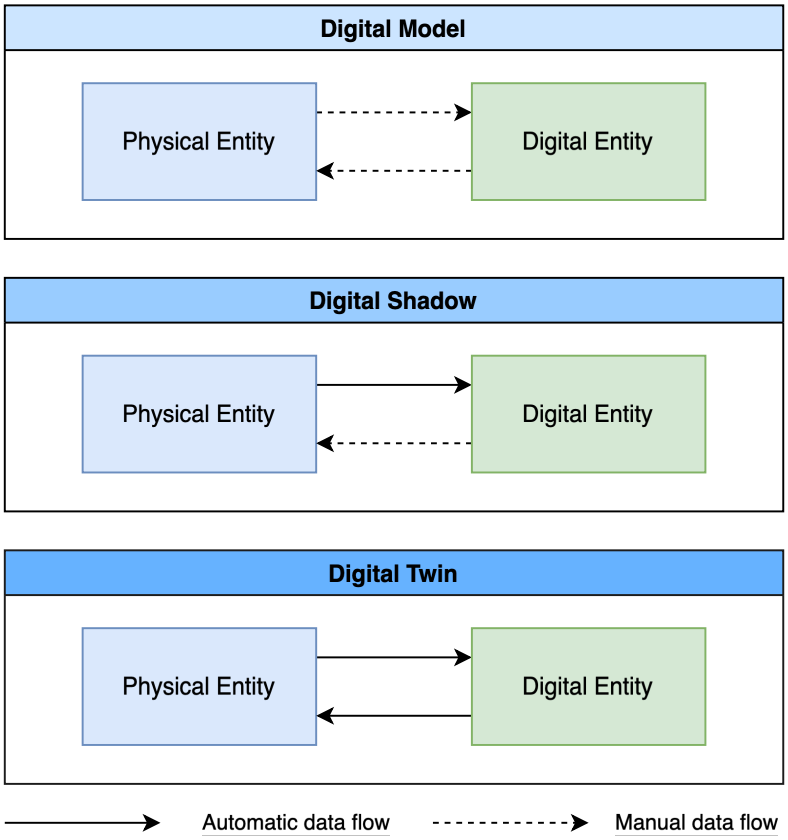

Figure 1.1: Data-flow relationships distinguishing Digital Model, Digital Shadow, and Digital Twin.

loop as humans are often required to interpret results and apply decisions back to the physical system (it is also known as *human-in-the-loop*). As a result, truly closed-loop digital twins, characterized by bi-directional data exchange and automated adaptation, remain scarce across domains (20, 21). This gap highlights the need for research that bridges the transition from monitoring-oriented twins to interactive, self-optimizing digital twin architectures.

In terms of implementation, closing the loop is challenging as calibration is required to ensure the simulator has the correct parameter values for simulation. Bridging this gap requires defining what trace schema and semantics are sufficient to represent execution in the digital continuum, and determining how simulator parameters can be tuned from observed traces to maintain adequate fidelity as workloads and configurations evolve. Without such calibration, what-if simulation results may diverge from real behavior, undermining trust in decision support.

Furthermore, even with a calibrated simulator, a digital twin provides limited opera-

tional value unless simulation outcomes can be transformed into concrete control actions. Simulator outputs typically take the form of predicted metrics or trajectories, whereas operators require actionable decisions such as selecting among candidate configurations or adjusting runtime parameters. A key open problem is therefore how to convert what-if results into policy choices that can be safely applied back to the physical system, forming a closed-loop workflow.

Finally, to claim that a closed-loop digital twin is effective, two forms of evidence are needed. First, the digital entity must demonstrate fidelity by reproducing key characteristics of observed execution within acceptable error bounds. Second, and most importantly, the closed loop must show decision-support value by demonstrating that simulator-informed actions should measurably improve performance and/or energy outcomes relative to baseline operational policies.

1.3 Research Questions

Based on the problem statement, we formulate the following research questions:

> **RQ1**: How can we design a closed-loop digital twin architecture for the digital continuum in which a trace-based simulation mirrors physical execution and drives physical-system adaptations via feedback?

RQ1 addresses the design question of how to realize a closed loop in the digital continuum, including how physical execution, the simulator, and control logic are coupled. It defines the system components and interfaces required for bi-directional information flow and feedback-driven adaptation.

> **RQ2**: How can a trace-based simulation workflow be operationalized to support closed-loop decision-making between a physical execution environment and its digital twin in the digital continuum?

RQ2 focuses on the implementation mechanisms needed to operate the closed loop end-to-end, from trace capture to simulation to actuation. It is decomposed into the following three sub-questions that separately address trace representation, simulator calibration, and decision translation.

> **RQ2a**: What trace schema and semantics are required to represent physical execution in a simulator-compatible form?

RQ2a investigates how to represent physical execution as structured trace data that preserves the information (such as CPU, memory, and energy usage) needed for simulation and

what-if analysis. It clarifies which events, metrics, and levels of granularity are necessary to produce simulator-compatible inputs.

RQ2b: How can the simulator be calibrated using observed traces so that it maintains sufficient fidelity over time?

RQ2b examines how to tune simulator parameters based on observed traces so that simulated behavior remains aligned with the physical system.

RQ2c: How can simulator outputs answer what-if questions and be converted into actionable control decisions for the physical system?

RQ2c focuses on translating simulation results into concrete actions, such as selecting among candidate operating configurations or adjusting runtime parameters that can then be applied to the physical system. It defines how what-if outcomes are mapped into implementable control decisions and how this action pathway completes the loop.

RQ3: What is the decision-support value of the closed loop? Do simulator-informed control actions improve performance and/or energy outcomes compared to baseline operational policies?

RQ3 evaluates whether closing the loop provides measurable benefits over baseline operation, using performance and energy as primary outcome dimensions. It also characterizes the operating conditions under which simulator-informed actions are reliable and beneficial.

1.4 Research Methodology

To answer the above research questions, the following research methodologies are adopted:

- **M1: Design, abstraction, and prototyping (22, 23, 24)**
  To answer **RQ1**, we survey and compare existing closed-loop and feedback-loop designs in digital twins and resource-management systems across domains. The survey is used to identify recurring responsibilities, information exchanges, decision stages, and constraints that must be addressed. We combine these findings with the challenges identified in the problem statement to derive requirements. We then organize the requirements into an architectural abstraction and refine it iteratively by checking whether the responsibilities are complete, whether their interfaces carry the required information, and whether the proposed feedback path can be instantiated in a prototype.

To answer **RQ2a**, we analyze the interface between a representative physical execution environment and a trace-driven simulator. We compare the observations that can be collected in practice with the information required for replay, fidelity assessment, what-if evaluation, and later decision making. The trace representation is refined by testing whether the prototype can carry the same observations from collection to simulator input without losing the information required by the later stages.

To answer **RQ2b**, we review calibration approaches for trace-driven simulation and define a repeatable process for selecting comparison targets, measuring disagreement between observed and simulated behavior, adjusting uncertain simulator parameters, and reassessing fidelity. The process is refined iteratively using evidence collected from the running prototype, which allows the calibration method to be evaluated against physical observations.

To answer **RQ2c**, we study how feedback-control and simulator-informed resource-management systems represent candidate changes and compare predicted outcomes. We use this analysis to define the abstractions and validation steps between simulation and actuation. These elements are then implemented and tested as an end-to-end prototype path, with each iteration used to identify missing information, ambiguous responsibilities, or unsupported actions.

- **M2: Experimental research (25, 26, 27)**
  To answer **RQ3**, we design controlled experiments around the running prototype. The experimental design defines the workload, baseline policies, controlled factors, repetition strategy, and outcome metrics before the alternatives are compared. The evaluation first determines whether the simulator reproduces the aspects of physical execution needed for the intended decision-support task. It then measures whether calibration changes the selected fidelity criteria. Finally, it compares simulator-informed control with baseline operational policies using performance and energy-related outcomes. Comparable settings use the same execution environment, workload-generation procedure, data-collection pipeline, and analysis method, while repetitions are used to characterize run-to-run variation.

- **M3: Open science, open-source software, and reproducible experiments (28, 29, 30, 31)**

To support the answers to **RQ1**–**RQ3**, we treat the implementation and experimental artifacts as part of the research method. The source code, workload definitions, experiment configurations, analysis scripts, and run outputs are versioned and organized so that each reported result can be traced to the procedure and inputs that produced it. Per-run evidence is retained separately from aggregated summaries, and the complete workflow is documented from environment preparation through execution and analysis.

1.5 Thesis Contributions

This thesis makes the following contributions:

- **C1 (Conceptual): Closed-loop digital twin architecture design for the digital continuum**
  We design a closed-loop digital twin architecture for the digital continuum, specifying the core components, interfaces, and data flows required to mirror physical execution and apply feedback-driven adaptations.

- **C2 (Conceptual): Trace-to-simulation-to-action methodology**
  We provide an end-to-end methodology for operating the closed loop, covering an execution-trace contract, a calibration method, and a decision-translation approach.

- **C3 (Experimental): Quantitative evaluation of baseline fidelity and decision-support value**
  We conduct a controlled quantitative evaluation of the proposed design and prototype. The evaluation examines simulator fidelity, the effect of calibration, and the decision-support value of simulator-informed control by comparing physical and simulated workload behavior, performance, and energy across different resource configurations and control strategies.

- **C4 (Artifact): Open-source prototype and reproducibility package**
  The prototype that implements the closed-loop digital twin is released as open-source software, together with scripts, configurations, and documentation to reproduce the full workflow from trace collection and conversion to simulation, calibration, what-if analysis, and actuation. Please refer to Appendix A for details of the reproducibility package.

1.6 Plagiarism Declaration

I confirm that this thesis work is my own work, is not copied from any other source (person, Internet, or machine), and has not been submitted elsewhere for assessment.

1.7 Thesis Structure

This thesis is organized as follows:

- Chapter 2 establishes the background and motivation by distinguishing digital models, digital shadows, and closed-loop digital twins, motivating simulation and calibration, and positioning execution traces as the interface between the physical and digital systems.
- Chapter 3 answers **RQ1** by deriving the operational workflow, requirements, and reference architecture.
- Chapter 4 answers **RQ2a**–**RQ2c** by realizing the operational trace contract, topology-aware calibration, and the state-bound path from what-if candidate evaluation to bounded Kubernetes actuation.
- Chapter 5 evaluates simulator fidelity and the effect of topology-aware calibration, then answers **RQ3** by comparing the physical runtime and RAPL-derived processor-domain energy outcomes of the closed-loop policies with the pending-Pod baseline. It also discusses threats to validity.
- Chapter 6 positions the proposed architecture and prototype relative to related work on digital-twin architectures, digital-entity realizations, state and result lineage, and calibration and fidelity assessment.
- Chapter 7 concludes the thesis by answering the research questions, summarizing the main findings and limitations, and identifying directions for future work.

2

Background and Motivation

This chapter establishes the conceptual background for the closed-loop digital twin developed in this thesis. It first distinguishes digital models, digital shadows, and closed-loop digital twins, motivating observability and controllability as requirements for feedback-driven operation (Section 2.1). It then argues for trace-based simulation as a practical virtual-twin mechanism that balances fidelity, cost, repeatability, and evaluation speed (Section 2.2). Because simulator fidelity can drift as workloads and infrastructure conditions change, calibration against observed executions is introduced as a necessary step (Section 2.3). Finally, execution traces are positioned as the interface contract connecting observation, simulation, calibration, and action generation (Section 2.4).

2.1 From Digital Shadows to Closed-Loop Digital Twins

Key takeaway. A closed-loop digital twin requires observability, controllability, and an end-to-end path from physical observation through what-if evaluation and decision making to actuation.

The definition of a digital twin has evolved since the term was introduced in the early 2000s. Early descriptions focused on three main elements: a physical entity, a digital entity, and the connection between them. Several works extended this view with additional dimensions, such as data and services, leading to a five-dimensional digital twin model (32, 33). However, the broader use of the term "digital twin" has also made the boundary between digital models, digital shadows, and digital twins less explicit (19). This distinction is important for this thesis because the objective is not only to represent a physical system digitally, but to support closed-loop decision making, which remains relatively underexplored in digital twin literature.

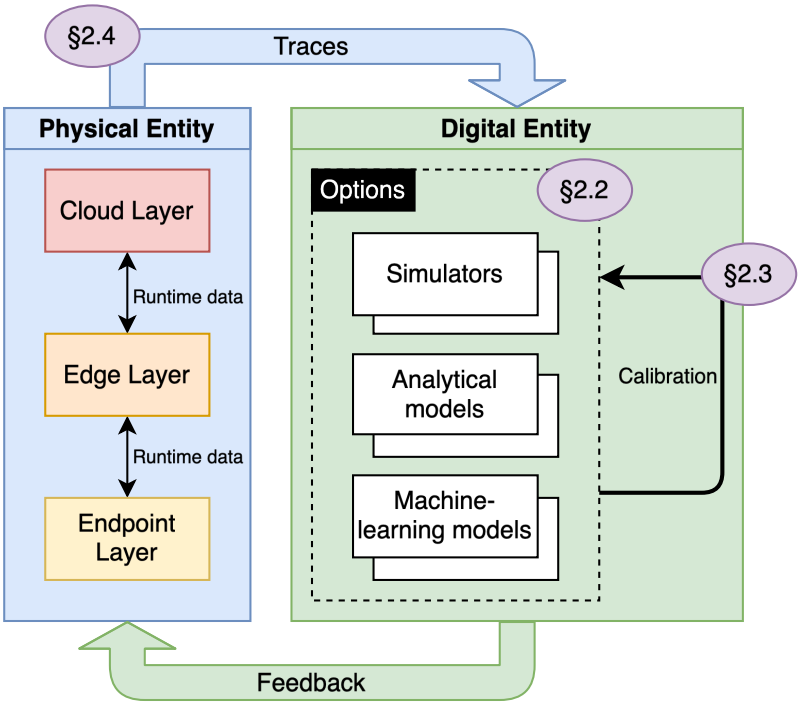

Figure 2.1: Centralized digital twin architecture for the digital continuum. The purple numbered callouts indicate how the following sections elaborate this architecture.

A digital model represents a physical system without automated data exchange, while a digital shadow can receive data from the physical system without automatically acting back on it. These forms are useful for modeling, monitoring, replay, and offline analysis, but they do not by themselves complete the feedback loop required for adaptation. A closed-loop digital twin requires a stronger form of coupling, in which the digital entity is not only updated from the physical system, but can also provide feedback that influences subsequent physical-system operation. The relevant gap for this thesis is not the absence of virtual representations, but the limited support for end-to-end architectures that connect observation, simulation, decision making, and actuation. Accordingly, this thesis adopts a closed-loop view of digital twins based on three core elements: a physical execution environment, a digital entity, and bidirectional data and feedback connections between them.

For a digital twin to support closed-loop operation, the physical entity must satisfy two fundamental interface requirements: **observability** and **controllability**. Observability refers to the ability to reconstruct, or sufficiently infer, the relevant state of a system from a limited set of measured variables, while controllability refers to the ability to influence this state through available control inputs (20, 34). In traditional engineering domains,

these requirements are often realized through sensors and actuators attached to a physical asset (35). In the digital continuum, however, the physical entity is not a single asset but a distributed execution environment composed of endpoint, edge, and cloud resources. Observability therefore requires interfaces that expose runtime information about the state and evolution of this environment, including workload execution, resource usage, performance, and energy-related behavior. Controllability refers to the ability to influence the execution environment through management interfaces, such as changing workload placement, modifying a deployment or resource configuration, scaling resources, or triggering a new execution.

At this stage, the digital entity can be treated as a black-box component, as illustrated in Figure 2.1. Its internal implementation is a design decision that is discussed in the following section (Section 2.2) in terms of model fidelity. Depending on the required fidelity and cost, the digital entity may be implemented as a simulator, an analytical model, a machine-learning model, or a combination of these approaches (36). The important architectural property is not the specific implementation, but the role it plays in the loop. It receives observations from the physical system, maintains a virtual representation of the relevant behavior, evaluates candidate decisions, and produces feedback that can be translated into actions on the physical side. In this way, the digital twin supports what-if analysis and decision making without applying every candidate action directly to the physical system.

More complex architectures are possible in the digital continuum. For example, a multi-layer digital twin can place separate digital entities at endpoint, edge, and cloud levels, as shown in Figure 2.2. Such architectures can improve timeliness, autonomy, and confidentiality by allowing some decisions to be made closer to the physical resources, while still enabling higher-layer analysis where more knowledge or computing power is available. Prior work follows this direction by distributing digital twins across continuum layers to improve scalability, response time, and privacy (37, 38). However, this distribution also introduces additional architectural complexity. Each local digital twin may only observe part of the system, such that communication, synchronization, and coordination between digital twins at different levels must be modeled explicitly.

This thesis adopts the simpler centralized digital twin abstraction shown in Figure 2.1. It considers a physical execution environment spanning the digital continuum layers. This abstraction is sufficient for studying the core feedback problem addressed in this thesis: how execution can be observed, represented as traces, simulated under alternative configurations, and converted into concrete control actions. Multi-layer digital twins remain an

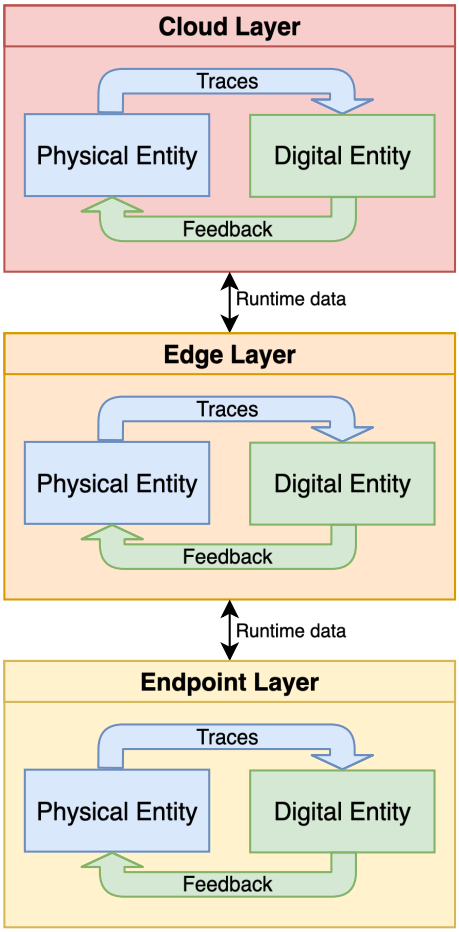

Figure 2.2: Multi-layer digital twin architecture for the digital continuum.

important extension, but they require additional mechanisms for synchronization, partial observability, and cross-layer coordination, which are outside the scope of this work.

2.2 Simulation as a Practical Digital Entity Mechanism

Key takeaway. Trace-based simulation provides a balance among fidelity, cost, evaluation speed, and repeatability across different digital entity mechanisms for what-if decision support.

A digital twin is useful for decision support when its virtual representation is faithful enough to the physical system it represents. This property is commonly referred to as *fidelity*, which describes how precisely the virtual representation mirrors physical reality in a measurable way. In the literature, fidelity is often related to the number of represented

	Physical execution	Simulation	Analytical model
Relative fidelity	High	Medium	Low–Medium
Relative cost	High	Medium	Low
Evaluation speed	Low	Medium–High	High
Repeatability	Low–Medium	High	High

Table 2.1: Qualitative trade-off between different approaches for evaluating system behavior (43, 44, 45).

parameters, their accuracy, and the level of abstraction used in the model (39, 40, 41). It is not about whether the digital entity resembles the physical entity in full detail, but whether it captures the states and key behaviors relevant to the intended use case.

Fidelity is especially important when the digital twin is used to evaluate future actions. If the virtual model does not reproduce the relevant behavior of the physical system with sufficient precision, what-if analysis may produce misleading predictions. In a closed-loop setting, this risk is amplified when recommendations from the virtual side are translated into concrete actions on the physical system. Low-fidelity results may therefore lead to poor decisions, causing deviations in system behavior or even system malfunctions (42).

However, the goal is not necessarily to maximize fidelity. Higher fidelity usually requires more measurements, finer-grained data, more detailed models, and stronger integration between the physical and virtual systems. This increases the cost of designing, implementing, and operating the digital twin. For this reason, prior work argues that a digital entity should target an optimum or fit-for-purpose fidelity level, where the benefit gained from additional detail justifies the cost required to obtain it (40).

This trade-off is closely related to abstraction. Any virtual representation must define which aspects of the physical system are represented and which aspects are ignored. A more detailed representation may preserve more physical behavior, but can make the model costly to construct, update, and execute. In contrast, a more abstract representation can improve efficiency, but may omit behavior that is important for the decision being evaluated (41, 42).

Table 2.1 summarizes this trade-off for three common ways of evaluating system behavior. Executing candidate actions directly on a production-equivalent physical testbed provides the highest empirical fidelity because behavior is observed from the real system or a close physical replica. However, this approach is costly and slow. Analytical models are usually faster and cheaper, but rely on stronger assumptions and may abstract away details that affect behavior in heterogeneous systems. Simulation provides a middle ground because

it abstracts from the physical system, but can still preserve enough behavior to support repeatable what-if analysis (46). This motivates the adoption of simulation in this thesis, specifically trace-based simulation, as the primary evaluation approach. The motivation for using traces and the corresponding requirements are discussed further in the subsequent section (Section 2.4).

2.3 Calibration for Simulation Fidelity

Key takeaway. Calibration uses physical evidence to adjust uncertain simulator parameters and quantify residual error, but the resulting fidelity remains specific to the evaluated metrics and operating conditions.

Using a simulator as the digital entity of a digital twin is not sufficient on its own. The predictions of a simulator depend on modeling assumptions, selected parameters, and the operating conditions under which those parameters were derived. As discussed in the previous section, fidelity is not a fixed property that can be assumed once a simulator has been implemented; it must be assessed against evidence from the physical system. Prior work on simulation fidelity describes this evidence as a fidelity referent: a structured representation of available real-world knowledge against which simulation realism can be compared (47). In this thesis, observed execution traces play this role by providing empirical evidence of how workloads, resources, and performance evolve in the physical execution environment.

Calibration is the process that seeks to align simulator output with this evidence. The literature defines calibration as refining uncertain simulator parameters by comparing simulator outputs with observed data, while distinguishing between simulator inputs and simulator parameters (48). The trace provides scenario information, such as task arrivals, execution times, resource usage, dependencies, and placement decisions, while calibration adjusts simulator parameters such as speed factors, communication costs, scheduling overheads, or energy-model coefficients. In other words, traces not only provide replay inputs but also supply the empirical basis for assessing whether the simulator reproduces the physical system with sufficient accuracy.

Calibration is especially important here because both workloads and infrastructure conditions change over time in the digital continuum. A parameter setting that reproduces one execution period may become inaccurate when workload intensity changes, hardware availability shifts, or network conditions vary. Without recalibration or repeated fidelity checking, simulation results can drift away from physical behavior, making what-if analysis

unreliable. Moreover, prior work argues that calibration should not be treated as simply finding one optimal parameter vector and then assuming that the simulator is correct. No parameter setting may match all observations, and residual discrepancy and uncertainty must be considered to avoid overestimating predictive accuracy (48). For this reason, calibration in this thesis should produce both calibrated parameters and measurable error bounds for the target metrics.

2.4 Execution Traces as the Interface Between Physical and Digital Systems

Key takeaway. Execution traces are semantic contracts that define the meaning and granularity of recorded events and preserve the minimum context required for replay, calibration, and what-if analysis.

Execution traces provide the practical interface between the physical execution environment and the digital entity. In a closed-loop digital twin, the physical system must expose enough information for the digital entity to reproduce, evaluate, and adapt execution, but it is not necessary to model the internal business logic of every task. Instead, the trace records the externally observable semantics that are relevant for simulation, such as task arrivals, start and end times, resource allocation, placement, dependencies, resource usage, failures, and energy-related measurements. In this sense, the trace acts as a contract between the physical and digital systems, where the physical system provides structured observations, and the simulator consumes those observations as replay and calibration input.

The trace interface should define not only the data format but also the semantics and granularity of recorded events. For example, a simulator-compatible trace should make clear how timestamps are interpreted, which events delimit task execution, how dependencies are represented, which resource measurements correspond to which execution interval, and which configuration was active during the observation period. Without these semantics, two systems may exchange trace files successfully while still disagreeing on what the trace means. The design of the trace schema is therefore an important part of the digital twin architecture, because it determines what behavior can be replayed, which parameters can be calibrated, and which what-if questions can be evaluated.

At the same time, the trace should not include all available monitoring data. Logging every observable event or metric can increase storage and processing overhead, complicate simulator input generation, and expose sensitive or proprietary information such as

user identities and application names. Prior work on trace archives therefore includes anonymization and information-reduction mechanisms to obscure sensitive information while retaining useful execution detail (49). Thus, the trace schema should preserve only the minimum information required for replay, calibration, and what-if analysis.

3

Design of Closed-Loop Digital Twin

This chapter presents the design of the trace-based closed-loop digital twin developed in this thesis. It first defines the design goal and adapts the MAPE-K model to an eight-step operational workflow that connects observation, trace publication, candidate generation, simulation, calibration, decision selection, and bounded actuation (Section 3.1). It then derives functional and non-functional requirements from this workflow, covering both the operations needed to complete the loop and the constraints under which it can be trusted (Section 3.2). Finally, it introduces a reference architecture spanning the physical space, a closed-loop control plane, and the digital space, and explains the responsibilities and interfaces of the components that realize the loop (Section 3.3).

3.1 Design Goal and Design Approach

The goal of this chapter is to answer **RQ1** by proposing a reference architecture for a trace-based closed-loop digital twin in the digital continuum. The design focuses on the architectural level, which defines the required logical components, the information exchanged between them, and the constraints that must hold to operate the loop.

The design approach is inspired by the **MAPE-K loop** from autonomic and self-adaptive systems. Autonomic computing aims to construct systems that can manage themselves according to high-level objectives provided by human administrators (50, 51). The MAPE-K model structures such self-management as a feedback loop consisting of Monitor, Analyze, Plan, and Execute functions operating over shared Knowledge (52). In this thesis, MAPE-K is used as a conceptual starting point for structuring the closed loop, while the concrete workflow is adapted to the needs of trace-based simulation and digital-twin operation.

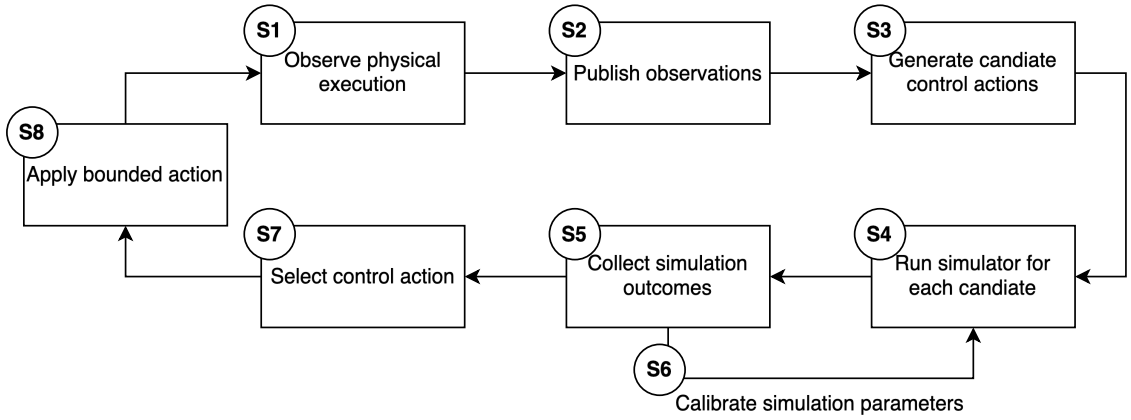

Figure 3.1: High-level operational workflow of one closed-loop iteration.

Instead of mapping each MAPE-K element one-to-one to a single component, the workflow further refines the general idea of monitoring, analyzing, planning, and executing into eight architectural responsibilities. As shown in Figure 3.1, one loop iteration starts by observing physical execution and publishing these observations as trace records. The system then generates candidate actions, evaluates them through simulation, collects the resulting outcomes, calibrates the simulator when needed, selects one control action, and applies only a bounded action to the physical system. The details of each step are as follows:

- **S1: Observe physical execution.**
  The system captures runtime data, such as workload execution and resource usage, from the physical execution environment.

- **S2: Publish observations.**
  Raw measurements are converted into structured traces. This makes physical observations consumable by later components without exposing them to platform-specific monitoring formats.

- **S3: Generate candidate control actions.**
  Based on the current system state, the system proposes a finite set of possible adaptations. These candidates define the decision space that will be evaluated by the simulator.

- **S4: Run the simulator for each candidate.**
  The simulator replays the trace-derived workload under each candidate configuration. This allows the system to estimate the effect of possible actions before applying them to the physical entity.

- **S5: Collect simulation outcomes.**
  The system collects the predicted outcomes produced by the simulator.

- **S6: Calibrate simulation parameters.**
  Simulator parameters are updated when simulated behavior no longer matches recent physical measurements with sufficient fidelity. Calibration keeps the virtual representation aligned with the physical system over time.

- **S7: Select control action.**
  A decision policy uses the predicted outcomes to rank the evaluated candidates and selects one as the decision.

- **S8: Apply bounded action.**
  The selected decision is translated into a supported physical-system action and applied back to the physical system.

This workflow emphasizes that a closed-loop digital twin is not merely a feedback arrow from the digital entity to the physical entity. It is a sequence of architectural responsibilities from observation, to proposal generation and evaluation, to decision selection, and finally to actuation.

3.2 Requirements Derived from the Closed Loop

This section derives the requirements from the closed-loop workflow introduced in Figure 3.1. The requirements make this workflow explicit by separating the functions that the architecture must provide from the quality constraints that must hold when these functions are connected.

3.2.1 Functional Requirements

The functional requirements (FRs) specify the behavior that the system must provide in order to complete one closed-loop iteration. Each requirement is derived from one or more steps of the workflow.

- **FR1: Observe the physical execution environment.**
  The system shall collect the runtime information needed to describe physical execution, including workload execution, resource state, configuration, timing, and energy-related behavior where available. This requirement supports **S1**.

- **FR2: Represent observations through a simulator-compatible trace interface.**
  The system shall convert physical observations into structured trace and system-state records with defined semantics, timestamps, event granularity, and configuration context. The trace interface shall preserve the information required for workload replay, calibration, and what-if evaluation. This requirement supports **S2**.

- **FR3: Generate candidate control actions from the current observed state.**
  The system shall produce a finite set of candidate control actions from the current system state. The candidate set defines the decision space to be evaluated by the digital side of the twin. This requirement supports **S3**.

- **FR4: Evaluate candidate control actions using a simulator-backed digital entity.**
  The system shall use trace-derived workload information and candidate configuration information to evaluate the expected outcome of each candidate. The evaluation shall produce predicted outcomes that can be compared by the decision policy. This requirement supports **S4** and **S5**.

- **FR5: Calibrate the digital entity using observed measurements.**
  The system shall compare simulated outputs with observed physical measurements and update simulator parameters when needed. Calibration shall use recent physical evidence so that the simulator remains aligned with the physical execution environment over time. This requirement supports **S6**.

- **FR6: Select a decision using an explicit decision policy.**
  The system shall rank or filter valid simulation outcomes according to stated objectives, such as minimizing runtime, minimizing power or energy consumption, maximizing utilization, or combining multiple objectives. The selected result shall remain an abstract decision rather than a platform-specific command. This requirement supports **S7**.

- **FR7: Apply the selected decision and continue the feedback loop.**
  The system shall translate the selected decision into a supported physical-system action and apply it through the actuation boundary. After actuation, the system shall observe the physical environment again so that the next loop iteration starts from the updated state. This requirement supports **S8**.

3.2.2 Non-Functional Requirements

The non-functional requirements (NFRs) define the quality properties that the closed loop must satisfy. They do not introduce new loop functions, but constrain how the functional requirements above should be implemented and connected.

- **NFR1: Fit-for-purpose fidelity.**
  The simulator-backed digital entity shall reproduce the characteristics of physical execution that are relevant to the target decision-support task. These characteristics may include workload progress, runtime or queueing behavior, resource usage, and power or energy trends. The goal is not to reproduce every low-level physical detail, but to preserve enough behavior for comparing candidate actions safely and meaningfully.

- **NFR2: Timely and state-valid loop execution.**
  Candidate generation, simulation, decision making, and actuation shall proceed only while the assumptions derived from the observed system-state snapshot remain valid. Each candidate and simulation outcome shall be associated with the system-state snapshot on which it was based. Before decision selection and again before actuation, the system shall verify that the current physical state remains compatible with the assumptions derived from that snapshot. If a material state change occurs, such as a change in resource availability or topology, simulation outcomes based on the affected snapshot shall be invalidated and shall not drive physical actions.

- **NFR3: Modularity.**
  Components shall communicate through explicit interfaces so that the components can be replaced with limited changes to the rest of the architecture.

- **NFR4: Reproducibility and traceability.**
  The system shall record the evidence needed to inspect and reproduce a closed-loop iteration.

- **NFR5: Bounded and safe actuation.**
  Control actions shall be restricted to operations supported by the physical execution environment. The architecture shall retain a no-operation path when evidence is incomplete, stale, too uncertain, or unfavorable compared with the current configuration.

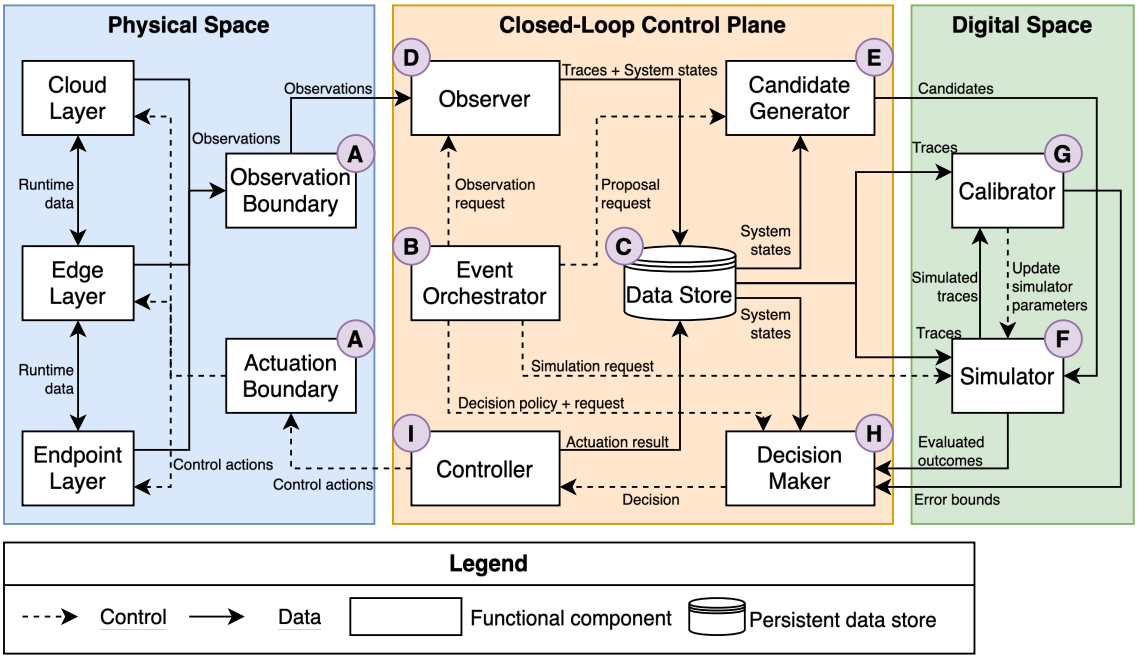

Figure 3.2: Reference architecture of the trace-based closed-loop digital twin. The purple callouts identify the architectural components discussed in Section 3.3.

Together, the functional requirements describe what the closed loop must do, while the non-functional requirements describe the constraints under which the loop can be trusted.

3.3 Closed-Loop Architecture

Figure 3.2 shows the reference architecture of the trace-based closed-loop digital twin. The architecture is divided into three logical spaces: the physical space, the closed-loop control plane, and the digital space. The physical space represents the execution environment across the digital continuum and is accessed only through explicit observation and actuation boundaries. The digital space contains the simulator and calibrator, which provide simulation and calibration evidence for decision support, but do not directly control the physical system. Between them, the closed-loop control plane coordinates the feedback loop. It manages loop state, stores evidence, generates candidates, selects decisions, validates result applicability against the current system state, and ensures that only valid decisions can become bounded actions. This separation keeps simulation, decision selection, and physical actuation as distinct responsibilities.

3.3.1 Design Concepts and Architectural Invariants

Before introducing the components, this section defines the main concepts used throughout the architecture.

System state. A system state is a snapshot of the physical execution environment at a given point in time. It includes information such as the active and available resources. Each state has a unique identifier so that candidates and simulation outcomes can be linked back to the observation from which they were derived.

Trace window. A trace window is the bounded part of recent workload and telemetry history used for simulation and calibration. It is assumed that recent behavior is more relevant to the next control decision than old behavior. Older observations may therefore be discarded or given less weight, which helps the loop remain responsive to current workload conditions.

Candidate. A candidate is a possible control decision together with the configuration that should be evaluated by the simulator. For example, a candidate may keep the current resource configuration, add or remove resources, or change a scheduling-related configuration. A candidate is not yet an applied physical action.

Simulation outcome. A simulation outcome is the predicted result of evaluating one candidate under the current trace window and simulator parameters. It may contain metrics such as predicted runtime, utilization, power, or energy consumption.

Decision. A decision is the candidate selected by the decision policy after simulation outcomes have been checked and ranked. The decision remains abstract. It states what should be changed, but it is not yet a platform-specific operation.

Bounded action. A bounded action is the concrete physical-system operation produced by translating a selected decision through the controller and actuation boundary. Only bounded actions can affect the physical execution environment.

No-operation candidate. The no-operation candidate represents keeping the current configuration. It provides a baseline for comparing other candidates and a safe fall-back when evidence is stale, incomplete, uncertain, or unfavorable.

The architecture follows four invariants. First, the simulator cannot apply actions directly to the physical execution environment. Second, every candidate and simulation outcome must be linked to the system state on which it was based. Third, the controller only

applies actions that are supported by the actuation boundary. Fourth, the no-operation path is always available so that the loop can avoid acting when the evidence is not strong enough.

3.3.2 Physical Observation and Actuation Boundaries

The physical execution environment boundaries define how the closed-loop control plane interacts with the physical side of the system. These boundaries correspond to callout (A) in Figure 3.2, which identifies the observation boundary and the actuation boundary.

The observation boundary exposes runtime information from the physical execution environment. This includes information such as workload execution, resource usage, topology, active resource capacity, and energy-related telemetry where available. The purpose of the observation boundary is not to expose every internal detail of the execution platform, but to provide the information needed to construct trace records and system-state snapshots. Through this interface, the architecture obtains evidence required by **FR1** and the fresh state information needed for the state-validity checks required by **NFR2**.

The actuation boundary exposes the supported management operations that the controller may apply. These operations may include changing resource availability, modifying placement or scheduling constraints, adjusting deployment-level resource limits, changing admission or scaling policies, or triggering another platform-supported reconfiguration. The exact operations depend on the physical execution environment, but the architectural requirement is that the control plane can only apply actions exposed through this boundary. This realizes **FR7**.

The separation between observation and actuation is important for safety. The simulator may evaluate possible actions, but it cannot directly control the physical execution environment. A physical action is applied only after the control plane has checked that the evidence is current, the decision is valid, and the action is supported by the actuation boundary. This enforces **NFR5**. These explicit boundaries also support **NFR3** by hiding platform-specific monitoring and actuation mechanisms behind stable interfaces.

3.3.3 Event Orchestrator

The event orchestrator, labeled (B) in Figure 3.2, coordinates the timing and state of the closed loop. It decides when a new loop iteration starts, requests a fresh system state, triggers candidate generation, submits simulation requests, and processes simulation

reports when they return. It acts as the coordination point for **FR1**–**FR7**, while the individual requirements are realized by the other architectural components correspondingly.

The orchestrator maintains the current system state and assigns identifiers to loop evidence. A simulation batch is based on a specific observed state. If the physical state changes while the simulator is still evaluating candidates, the resulting simulation report may no longer describe the current situation. The orchestrator therefore rejects results that are too old, incomplete, or based on a state that no longer matches the current observed state. This realizes the state-validity constraint in **NFR2**.

3.3.4 Data Store

The data store, labeled (C) in Figure 3.2, provides the shared evidence base for the loop. At the architectural level, it must retain four categories of evidence: (1) the observed state and trace window used as input, (2) the candidate set and simulation report produced from that state, (3) the calibration or fidelity evidence used to interpret the report, and (4) the selected decision together with the actuation result. These records provide the traceability required by **NFR4**. The data store also supports **NFR3** because components exchange structured records instead of depending on each other's internal implementation.

Conceptually, the data store plays the "knowledge" role of the control loop. It provides the evidence used by the observer, candidate generator, simulator, calibrator, decision maker, and the controller, while leaving the decision logic to the components that consume this evidence.

3.3.5 Observer

The observer, labeled (D) in Figure 3.2, turns physical execution into information usable by the closed-loop control plane. It receives measurements through the observation boundary and publishes workload/resource observations together with system-state snapshots that describe the current physical configuration. By doing so, it provides the observation and trace-interface functions required by **FR1** and **FR2**.

At the architectural level, the observer translates platform-specific monitoring data into simulator-facing evidence. It bridges the physical execution environment and the control plane by converting raw measurements into observations and system-state records with explicit timestamps, source context, and state identifiers. The design only assumes that the observer provides enough information for later simulation, calibration, and decision making. The concrete trace schema and its semantics are discussed in Section 4.2.

The observer should operate asynchronously with respect to the decision loop. Physical execution and monitoring continue while candidates are generated, simulated, and selected. This avoids blocking observation during long simulations and allows the next loop iteration to start from fresh evidence.

The observer supports **NFR2** by publishing timestamped system-state snapshots with state identifiers needed by later components to determine whether simulation evidence remains valid. It also supports **NFR3** through a monitoring-independent output interface, and **NFR4** through traceable observation records.

3.3.6 Candidate Generator

The candidate generator, labeled (E) in Figure 3.2, defines the decision space for one loop iteration. It receives the current system state and produces a finite set of candidate control actions for simulation-based evaluation. Candidate generation is domain specific: in a resource-management setting, candidates may change available capacity, placement constraints, scheduling policies, or other supported configuration options. This component provides the candidate-generation function required by **FR3**.

At the architectural level, candidates represent possible decisions rather than simulator-specific requests or platform-specific commands. This keeps candidate generation modular because the generator only enumerates the alternatives that should be considered for the current observed state. It does not need to know how the simulator encodes those alternatives internally, and it does not decide how an accepted decision will later be enacted by the physical execution platform. The simulator consumes the candidate configuration to estimate its outcome, while actuation is reached only after the decision maker has selected one abstract decision. As a result, the same candidate-generation logic can be reused when the simulator backend, decision policy, or actuation backend changes. This supports **NFR3**.

Each candidate should be linked to the system state on which it was based. This linkage allows later components to reject outcomes that no longer match the current physical state, supporting **NFR2**.

The candidate set should always include a no-operation candidate. This candidate represents keeping the current configuration, provides a baseline for comparison, and gives the loop a safe fallback when no proposed change is sufficiently supported by the available evidence. This fallback contributes to **NFR5**.

3.3.7 Simulator and What-If Evaluation

The simulator, labeled (F) in Figure 3.2, is the main mechanism used by the digital space to answer what-if questions. It receives the trace window, candidate configurations, and simulator parameters, then estimates the expected outcome of each candidate. The purpose of this step is to compare possible future configurations without applying each one to the physical execution environment. This component realizes the evaluation function of **FR4**.

Each candidate should be evaluated under comparable conditions. In particular, candidates from the same decision cycle should use the same trace window and compatible simulator parameters. This allows the decision maker to interpret differences in predicted runtime, power, energy, or utilization as consequences of the candidate configuration rather than consequences of different input workloads.

The simulator produces evidence, not commands. The simulator does not select the final decision and does not apply actions to the physical execution environment. This supports **NFR5** by preventing simulation output from bypassing the control plane.

The simulator output should preserve only the metadata needed to validate and inspect the result. The most important fields are the state identifier, the trace window, the evaluated candidate, the simulator-parameter version, and the produced metrics. The state identifier and trace window indicate which physical situation was simulated. The evaluated candidate and produced metrics show what was compared by the decision maker. The simulator-parameter version records which calibrated simulator configuration produced the result. This supports **NFR2** and **NFR4**.

3.3.8 Calibrator and Fidelity Evidence

The calibrator, labeled (G) in Figure 3.2, is the component responsible for maintaining the fidelity of the simulator-backed digital entity. It realizes **FR5** by comparing simulated behavior with recent physical observations and updating simulator parameters when the difference affects decision-critical characteristics. This also keeps the simulator aligned with the physical execution environment for the metrics required by **NFR1**.

At the architectural level, calibration provides evidence about whether simulator outputs can be trusted for decision making. The calibrator exposes updated simulator parameters and error bounds to the simulator and decision maker, respectively. These outputs make fidelity explicit for **NFR1** and record the calibration evidence required by **NFR4**.

A calibration result is only valid within the context in which it was obtained. Therefore, the calibrator should report not only the updated simulator parameters, but also the

validity context of those parameters, including the trace window and the observed system state. This matters when the physical topology changes, because parameters calibrated for one topology may not be reliable for another. The concrete calibration procedure, including the parameter search strategy and the handling of topology changes, is part of the implementation described in Section 4.3.

3.3.9 Decision Maker and Policy-Based Selection

The decision maker, labeled (H) in Figure 3.2, converts evaluated simulation outcomes into one selected decision. It receives candidate outcomes from the simulator, the current observed system state, and the active decision policy from human administrators. This component satisfies **FR6** by making candidate selection explicit and policy-driven.

The current observed system state is passed to the decision maker because the simulator may have an incomplete view of the live physical execution environment. For example, the simulator can evaluate candidates using the trace window and candidate configurations it receives, while the live system may also contain pending jobs or platform-side constraints that were absent from the simulation input. Providing the current state lets the decision maker reason over both predicted outcomes and live physical context before selecting an action.

The decision policy defines the objective used to compare candidates. For example, the policy may minimize predicted runtime, minimize predicted power or energy consumption, maximize utilization, or combine multiple objectives using a weighted or ranked policy. Before ranking candidates, the decision maker verifies that each outcome remains valid for the current system state. Outcomes are rejected when they are stale, incomplete, or no longer compatible with the current state, which supports **NFR2**.

The decision maker should also check the fidelity evidence associated with the simulation. Outcomes whose error bounds or uncertainty exceed the policy's tolerance are rejected or treated conservatively. This connects candidate selection to **NFR1**.

The no-operation candidate is treated as the comparison baseline. A candidate should only be selected when its predicted improvement is meaningful under the current policy and trustworthy under the current fidelity evidence. If all candidates are unsafe, stale, incomplete, or worse than the no-operation baseline, the decision maker selects no-operation. This fallback supports **NFR5**.

The decision maker emits an abstract, platform-independent decision, leaving its translation into a physical operation to the controller. This boundary decouples decision selection from platform-specific actuation. The decision policy can change without modifying the

controller interface, while a different physical backend can be supported by adapting or replacing the controller without changing the simulator or decision logic. This separation supports **NFR3**. The selected decision and its supporting evidence are recorded for **NFR4**.

3.3.10 Controller and Bounded Actuation

The controller, labeled (I) in Figure 3.2, is the component that applies selected decisions to the physical execution environment. It receives an abstract decision from the decision maker, translates it into a supported physical-system operation, and applies that operation through the actuation boundary. For example, a decision to change the number of active machines may be translated into platform-specific operations that make selected machines available or unavailable for scheduling. This translation and application path implements **FR7**.

The controller is responsible only for enacting selected decisions and does not evaluate candidates or determine which action should be selected. Platform-specific actuation logic is therefore localized within this component, keeping it separate from simulation and decision making. This separation supports **NFR3**.

The controller must treat actuation as bounded and observable. It should apply only supported operations, record whether the action succeeded or failed, and trigger re-observation after actuation. These records and bounds address **NFR5** and **NFR4**. After actuation, the next loop iteration starts from the newly observed physical state, so the measured physical state remains the source of truth.

3.4 Summary of Design

This chapter presented the design of a trace-based closed-loop digital twin for the digital continuum. The architecture connects physical observation, trace-based simulation, calibration, decision making, and bounded actuation. The design organized the functional components into three conceptual spaces with explicit interfaces separating physical execution from simulation and control.

During each loop iteration, the observer captures the physical execution and produces structured traces. The candidate generator derives possible actions, which are evaluated by the simulator using the trace-derived workload. The calibrator maintains simulation fidelity, while the decision maker selects a decision based on predicted outcomes, live system

Component	Main FR coverage	Main NFR coverage
(A) Physical execution environment boundaries	FR1, FR7	NFR2, NFR3, NFR5
(B) Event orchestrator	Coordinates FR1–FR7	NFR2
(C) Data store	/	NFR3, NFR4
(D) Observer	FR1, FR2	NFR2, NFR3, NFR4
(E) Candidate generator	FR3	NFR2, NFR3, NFR5
(F) Simulator	FR4	NFR2, NFR4, NFR5
(G) Calibrator	FR5	NFR1, NFR4
(H) Decision maker	FR6	NFR1, NFR2, NFR3, NFR4, NFR5
(I) Controller	FR7	NFR3, NFR4, NFR5

Table 3.1: Requirement coverage of the closed-loop architecture.

state, and available fidelity evidence. The controller then translates the selected decision into a supported physical-system operation, completing the feedback loop.

The event orchestrator and data store coordinate this process and preserve the artifacts of each iteration, including observations, candidate configurations, simulation results, decisions, and actuation outcomes. State-validity checks, fidelity validation, bounded actuation, and a no-operation fallback prevent outdated or insufficient simulation evidence from causing unsafe adaptations.

Table 3.1 summarizes how the architectural components address the functional and non-functional requirements defined in Section 3.2. Together, these components establish the design foundation for the trace-to-simulation-to-action workflow. The following chapter describes how this architecture is instantiated using concrete technologies and implementation mechanisms.

4

Realization of Closed-Loop Digital Twin

This chapter presents the prototype realization of the trace-to-simulation-to-action workflow introduced in Chapter 3. It first describes the simulation and framework basis, the supporting technologies, and the assumptions that bound the prototype (Section 4.1). It then addresses **RQ2a** by defining an operational trace contract for workload execution, physical telemetry, system snapshot, and state lineage (Section 4.2). Next, it addresses **RQ2b** through window-based calibration of the OpenDC CPU power model against recent processor-domain power measurements, with each estimate scoped to the topology that produced the evidence (Section 4.3). Finally, it addresses **RQ2c** by generating state-bound what-if candidates, validating asynchronous simulation reports, selecting an abstract decision under an explicit policy, and translating that decision into a bounded Kubernetes node-schedulability action followed by re-observation (Section 4.4). These mechanisms instantiate the closed-loop path from physical observation to simulator-informed action.

4.1 Implementation Basis and Prototype Scope

The implementation follows the three logical spaces introduced in Chapter 3. Continuum and Kubernetes realize the physical space, while OpenDC serves as the simulation engine at the core of the digital space. The observation, communication, decision-making, and actuation components form the control plane that connects the physical and digital spaces. The following subsections explain how these spaces are realized and define the scope of the prototype.

4.1.1 Simulation and Framework Basis

The simulation engine of the digital entity is implemented using **OpenDC**. It is an open-source platform for modeling and simulating cloud datacenters through discrete-event simulation (53). It represents datacenter infrastructure, computational workloads, resource management, and scheduling as models whose behavior progresses through a sequence of timestamped events. A simulation is supplied with a topology describing the available machines and a workload describing the submitted computational tasks. It can then report performance and resource-consumption metrics, including task completion times and estimated energy and power consumption.

OpenDC focuses on cloud-datacenter simulation and does not represent every layer and interaction of the digital continuum. It was selected not as a complete model of the continuum, but as a concrete means of demonstrating the feasibility of the closed-loop design proposed in Chapter 3. The implementation must show that observations from a physical execution environment can be translated into simulator-compatible input, that alternative resource configurations can be evaluated, and that the resulting evidence can support a bounded physical action. OpenDC provides the trace-driven simulation capabilities needed to realize and study this path at the cloud-resource level. This limited scope is sufficient to examine the decision-support value of the closed loop, while support for additional continuum layers would require other or extended simulation models.

The thesis prototype further builds on **OpenDT**, an open-source datacenter digital shadow framework that uses OpenDC as its simulator (54). OpenDT combines historical trace replay, trace-driven simulation, and the calibration of simulator parameters against physical measurements. Its broader framework envisions feedback to physical infrastructure under human supervision, but automated steering is not implemented in the evaluated prototype. OpenDT therefore provides the trace-processing, simulation, and calibration foundation from which the closed-loop prototype in this thesis is developed.

This thesis extends that foundation by connecting OpenDT to a live Kubernetes execution environment and implementing the path from observation to physical action. The extension introduces Kubernetes-specific observation of workload execution, resource usage, power, and topology. It also constructs candidate topologies from observed physical state, simulates and selects among alternative candidates according to the configured policy, and applies bounded actions through the Kubernetes API. Table 4.1 summarizes this relationship.

Aspect	OpenDT foundation	Extension in this thesis
Physical input	Historical workload and energy traces from SURF	Traces produced from live Kubernetes execution
Simulation	OpenDC replay of recorded workloads	Multiple OpenDC simulations evaluating alternative candidates
Calibration	Window-based comparison of simulated output using historical physical telemetry	Calibration using recent physical evidence and the currently observed topology
Decision support	Simulation and calibration results exposed for human-supervised analysis and feedback	Policy-based comparison and automatic selection of candidate outcomes
Physical action	Human-supervised feedback. Automated steering not implemented	Bounded actions applied automatically through the Kubernetes API

Table 4.1: OpenDT foundations and extensions implemented in this thesis.

4.1.2 Physical Environment

The physical execution environment is deployed using **Continuum**, a framework for automating infrastructure deployment and benchmarking across cloud, edge, and endpoint resources (55). Continuum creates the virtualized testbed and installs Kubernetes as its resource manager. **Kubernetes** executes the experimental workloads as finite Jobs and exposes the node, Pod, and Job information required for trace generation (56). The Kubernetes API server also provides the actuation interface through which the prototype applies supported actions to the physical environment.

The prototype focuses on the following what-if question: *If the number of worker nodes available for Kubernetes scheduling is changed, how does this affect workload runtime and power consumption?* This question restricts the implemented decision space to changing the amount of active scheduling capacity. For simplicity, the prototype does not provision or physically remove machines. Instead, its actuation interface controls which existing worker nodes are available to Kubernetes for scheduling.

4.1.3 Supporting Technologies and Deployment

The principal technologies used by the prototype are summarized in Table 4.2. They support workload execution, observation, communication, simulation, persistence, deployment, and inspection.

Docker Compose deploys the main prototype components as separate services. The `k8s-observer` service collects workload execution, resource usage, and power measure-

Technology	Role in the prototype
Kubernetes (56)	Orchestrates the finite Job workloads, exposes workload and node state, and provides the API for changing node schedulability.
Prometheus (57)	Stores and queries sampled time-series measurements exposed by the physical environment and monitoring components.
Scaphandre (58)	Reads processor energy counters exposed through RAPL and exports power-related measurements. These measurements provide physical evidence for calibration.
OpenDC (53)	Executes trace-driven, discrete-event simulations of the workload under each candidate topology and reports predicted performance and energy metrics.
Kafka (59)	Transports typed records asynchronously between the observer, calibrator, simulator, and control components.
PostgreSQL (60)	Persists observed workload, resource-usage, and node-power records for structured queries.
Apache Parquet (61)	Stores simulation outputs and exported experiment records in a column-oriented format for further offline analysis.
Docker Compose (62)	Defines and coordinates the infrastructure and application containers used by the closed-loop prototype.
FastAPI (63)	Exposes programmatic interfaces for accessing selected prototype data and operations.
Grafana (64)	Provides the human-facing interface for understanding physical and simulated measurements, available capacity, and the actions taken by the prototype.

Table 4.2: Principal technologies and their roles in the prototype.

ments from the physical environment. The `simulator` service invokes OpenDC to evaluate the observed workload and candidate topologies, while the `calibrator` service compares simulated output against recent physical measurements and publishes calibrated parameters for subsequent simulations. The `orchestrator` service coordinates observation of the current Kubernetes state, candidate generation, simulation requests, policy-based selection, and actuation. **Kafka** and **PostgreSQL** provide shared communication and persistence services. A **FastAPI** service exposes selected measurements, simulation outputs, and control records, which **Grafana** queries to provide a human-readable view of the physical system and the operation of the closed loop. Separating these responsibilities into independently deployable services contributes to satisfying the modularity non-functional requirement (**NFR3**) as defined in Chapter 3.

4.1.4 Prototype Scope and Assumptions

Workload scope. The prototype focuses on finite Kubernetes Jobs instead of continuously running services. A Job can be translated into a computational task with a submission time and one or more demand fragments, which corresponds to the trace-driven workload representation accepted by OpenDC. The workload generator used for the evaluation follows this task-based model. Its arrival patterns and experimental configurations are described in Chapter 5.

Control scope. The implemented decision variable is the number of schedulable worker nodes. Increasing the target count makes additional existing workers available to the Kubernetes scheduler, while decreasing it marks excess workers as unschedulable. Marking a worker as unschedulable prevents the placement of new Pods on that node, but it does not evict or migrate Pods that are already executing there. The implemented action therefore changes future workload placement within the bounded capacity of the existing cluster rather than immediately redistributing running workloads. It does not physically provision, start, stop, or destroy machines. The prototype also does not control processor frequency or resize individual Pods.

Infrastructure assumptions. The prototype assumes that all worker nodes operate at the same CPU frequency and can be represented using the same CPU power model. During an experiment, changes in the RAPL-derived power measurements collected through Scaphandre are assumed to result primarily from the experimental workload. Measurements from the worker nodes are also assumed to be sufficiently comparable to permit their aggregation.

Previous benchmark results show that RAPL readings respond to CPU workloads and reflect changes in CPU utilization, supporting their use as workload-sensitive physical evidence for calibrating a CPU power model (65). However, RAPL covers only processor-related domains and its readings may fluctuate. The aggregated measurements from the active nodes are used only to calibrate the simulator's CPU power model and are not interpreted as the total power consumption of the machines.

Calibration scope. The implemented calibration in this prototype is limited to OpenDC's CPU power model. It adjusts a selected power-model parameter by comparing simulated power with recent RAPL-derived physical measurements. CPU performance, workload, scheduling, and resource-sharing parameters are not calibrated. The calibration therefore

aims to improve the alignment between simulated and observed processor-domain power instead of the accuracy of every simulator output.

Simulation boundary. OpenDC models the workload demand and infrastructure properties required to compare candidate topologies, but it does not reproduce every internal behavior of Kubernetes. Scheduler constraints, Pod-pending behavior, control-plane delays, and other platform effects may therefore appear differently in simulation and physical execution. Simulation outcomes are consequently treated as evidence for comparing candidates rather than exact predictions.

4.2 Operational Trace Contract

This section answers **RQ2a**: *What trace schema and semantics are required to represent physical execution in a simulator-compatible form?*

Answer to RQ2a. RQ2a is addressed through an operational contract that combines simulator-facing execution records with closed-loop lineage. **Workload records** preserve task arrival, duration, declared CPU and memory capacity, and sampled CPU demand for OpenDC replay. **Telemetry records** preserve timestamped processor-domain energy increments and interval-average power for calibration, while **system snapshots** preserve the active topology and the capacity bound used to construct what-if candidates. State, batch, candidate, and report metadata retain the **lineage** needed to determine whether asynchronously returned outcomes remain applicable to the current physical state. The workload, telemetry, and system snapshot records provide simulation and calibration input, whereas lineage supports state-valid closed-loop decision making.

Realizing this contract requires bridging the abstraction mismatch between Kubernetes and OpenDC. Kubernetes exposes Jobs, Pods, resource declarations, and lifecycle events, while the observability components provide sampled resource-use measurements. OpenDC, in contrast, consumes timestamped computational tasks, demand fragments, and a topology of simulated machines. The contract therefore specifies which properties cross the physical–digital boundary and how their timestamps, units, observation granularity, and configuration context are interpreted. Following the minimum-information principle introduced in Section 2.4, it retains only the information required for workload replay, calibration, what-if analysis, and state-valid decision making.

The intermediate representation also localizes platform-specific translation. The observer converts Kubernetes observations into the contract. OpenDC consumes the workload and

system snapshot records, the calibrator uses telemetry as physical evidence, and the control plane uses lineage to validate returned outcomes. A change in the physical platform requires only adaptation of the observer-side translation, while replacing the simulator requires only adaptation of the simulator-facing mapping, provided that both preserve the operational contract. The contract may still evolve when either side requires additional information, but such changes remain localized at the interface. Together, these records and semantics realize **FR2**.

4.2.1 Workload Representation

The workload trace translates completed Kubernetes workloads into the task–fragment representation consumed by OpenDC. The observer monitors Jobs and their Pods in the configured namespace and records their creation, start, and finish times. It also reads the CPU and memory limits declared for the workload container in the Kubernetes YAML specification. Separate resource-usage workers sample CPU and memory use while the workload executes. When the workload reaches a terminal state, the observer joins its lifecycle information, declared limits, and captured samples to produce one `Task` with zero or more `Fragment` records.

Creating the task after termination ensures that its execution interval is known before it enters the simulator-facing trace. The trace therefore represents completed execution rather than the instantaneous set of running or pending workloads. Pending workloads remain visible through the live Kubernetes state, but they do not become OpenDC tasks until sufficient execution information is available.

Table 4.3 summarizes the main workload fields. The submission time is derived from the Kubernetes creation timestamp such that workload arrival is preserved. The task duration is calculated from the recorded start and finish times. CPU and memory capacities originate from the limits declared under the container resource specification in the workload YAML. The sampled CPU use is represented separately by the fragments.

Some observed information is deliberately omitted from the exchanged trace. Although the original workload placement is captured by the observer, it is not exchanged as OpenDC recomputes placement for each simulated topology. Failed executions are also captured but not represented as distinct simulator events. Kubernetes-specific and application-specific identifiers are also excluded because they do not affect simulation at the chosen abstraction level. These omissions also reduce unnecessary coupling to Kubernetes and avoid disclosing application-specific names.

Field	Type/unit	Implemented semantics
`Task.id`	Integer	Unique task identifier and parent key for its fragments.
`Task.submission_time`	UTC timestamp	Kubernetes creation time representing workload arrival.
`Task.duration`	Milliseconds	Difference between recorded finish and start times.
`Task.cpu_count`	CPU cores	CPU limit declared in the workload container specification.
`Task.cpu_capacity`	MHz	Simulator capacity derived from the declared CPU limit and configured processor frequency.
`Task.mem_capacity`	MB	Memory limit declared in the workload container specification.
`Fragment.task_id`	Integer	Reference to the parent task.
`Fragment.duration`	Milliseconds	Time from the preceding capture point to the current sample.
`Fragment.cpu_count`	CPU cores	CPU allocation associated with the parent task.
`Fragment.cpu_usage`	MHz	Sampled CPU use converted to simulator demand using the configured frequency.

Table 4.3: Fields and semantics of the workload trace.

4.2.2 Physical Telemetry

Workload records describe computational demand, whereas calibration requires physical measurements that provide ground truth for assessing and adjusting simulated behavior. The prototype therefore records the selected calibration target in a separate telemetry trace.

In the current prototype, the selected target is processor-domain power. Since the worker nodes are distributed, the prototype configures a separate Scaphandre source for each node. The observer periodically retrieves the cumulative RAPL energy counter from each source and aggregates the derived node-level measurements into timestamped `Consumption` records. The contract could be extended with observations for other calibration targets, but such targets are not implemented in the current prototype.

RAPL exposes cumulative energy rather than instantaneous power. The first successful reading for each source establishes a baseline and does not produce interval evidence. For every subsequent reading, the observer normally subtracts the preceding counter value from the current value. If the current value is smaller, the observer assumes one counter wrap-

Field	Type/unit	Implemented semantics
`Consumption.timestamp`	UTC timestamp	Capture time marking the end of the measured interval.
`Consumption.energy_usage`	Joules	Aggregated energy increment since the preceding successful capture.
`Consumption.power_draw`	Watts	Aggregated interval-average power derived from the node-level energy increments and elapsed intervals.

Table 4.4: Fields and semantics of the physical telemetry trace.

around and adds the energy remaining before the maximum counter range to the current reading (66). This correction assumes that no more than one wrap-around occurs between consecutive readings and that all the configured sources use the same counter range. The resulting energy difference is then converted from microjoules to joules. Dividing this interval energy by the elapsed time between the two readings yields the source's average power in watts.

Table 4.4 distinguishes the capture timestamp from the two quantities derived from the cumulative counters. The `energy_usage` field represents an energy increment rather than a cumulative counter value. The `power_draw` field represents average power over the observed intervals instead of instantaneous power. The timestamp provides the temporal key used to select physical readings from the recent evidence window and to align them with the corresponding OpenDC power output.

4.2.3 System Snapshot

Workload and telemetry observations must be interpreted together with the resource configuration under which they were produced. The same workload may exhibit different runtime and power behavior when executed with different numbers or types of machines. Therefore, the prototype records the observed resource configuration in a `SystemSnapshot` that provides the physical baseline required to interpret recent observations and construct alternative what-if configurations.

A `SystemSnapshot` combines four elements of the observed system state: a `state_id`, a timestamp, a `TopologySnapshot`, and additional state that constrains the available decision space. In the current prototype, this additional state is `max_available_node_count`. The `TopologySnapshot` describes the machines currently represented as active in the physical environment. It provides the simulator-facing representation of the physical baseline used for workload replay, calibration, and the construction of candidate configurations. In

Field	Type/unit	Implemented semantics
`SystemSnapshot.state_id`	UUID string	Unique identifier of the observed system state snapshot.
`SystemSnapshot.observed_at`	UTC timestamp	Time at which the represented resource configuration was observed.
`SystemSnapshot.max_available_node_count`	Integer	Upper bound on the existing machine capacity that candidate generation may request.
`SystemSnapshot.topology_snapshot`	`TopologySnapshot`	Simulator-facing description of the currently active resource configuration.
`TopologySnapshot.topology`	OpenDC topology	Collection of grouped host descriptions used as the physical baseline for simulation.
`Topology.Host.count`	Integer	Number of active machines represented by the grouped host description.
`Topology.Host.cpu`	CPU cores, MHz	Processor capacity represented through the core count and configured frequency.
`Topology.Host.memory`	Bytes	Allocatable memory represented in the simulated topology.
`Topology.Host.cpuPowerModel`	OpenDC PowerModel	CPU power model used to estimate processor-domain power draw.

Table 4.5: Fields and semantics of the system snapshot records.

contrast, `max_available_node_count` records the upper bound on the number of existing machines that may be made available to the workload. The topology therefore describes the configuration currently in use, while the maximum available node count defines the boundary within which candidate configurations may be generated.

Within the `TopologySnapshot`, the observed configuration is translated into the host representation expected by OpenDC. Workers with the same node type, allocatable CPU count, and allocatable memory are grouped into one OpenDC host description. The host count records how many active machines are represented by that description. Each host also carries a CPU power model used to estimate processor-domain power. Together, these fields define the capacity and power characteristics under which the trace-derived workload is simulated.

The timestamp records when the complete `SystemSnapshot` was observed, while the `state_id` gives that snapshot a unique identity across component boundaries. A new identifier is assigned when either the active topology or its available-capacity bound changes.

Candidates and simulation reports retain this identifier, allowing the control plane to reject evidence that no longer corresponds to the current resource baseline. The following subsection (Section 4.2.4) describes how this identity is propagated through candidate and result records.

Table 4.5 summarizes the fields that describe the active simulation baseline, its observation context, and the capacity boundary for candidate generation.

4.2.4 State, Candidate, and Result Lineage

Workload, telemetry, and system snapshot records provide the simulation input, calibration evidence, and physical baseline required for what-if evaluation. A closed loop must additionally determine whether returned outcomes remain applicable to the physical state currently being observed. Candidate evaluation proceeds asynchronously, and the topology or available-capacity boundary may change while the simulator is evaluating a candidate set. A result therefore cannot be interpreted independently of the observed state from which its candidate was generated. The prototype preserves this relationship through lineage metadata carried across the state, candidate, and result records.

Lineage begins with the `state_id` assigned to the observed `SystemSnapshot`. When candidate generation is triggered, the resulting `Candidate`s are grouped into a `SimulationBatch`. Each candidate receives a `candidate_id` that distinguishes it from the other alternatives in the same batch. A batch carries a `batch_id` and a `based_on_state_id`, which identifies the physical baseline from which the candidate configurations were constructed. After evaluation, the simulator returns a `SimulationBatchReport` that repeats the `batch_id` and `based_on_state_id`. Within this report, each `CandidateOutcome` retains the `candidate_id` of the evaluated candidate. The report also records a `created_at` timestamp that provides temporal context for later freshness validation.

These identifiers allow the control plane to establish whether a returned report belongs to a known candidate batch, associate each outcome with its original candidate, and determine which observed physical state the evaluation assumes. The timestamp additionally allows the age of the returned evidence to be assessed. These fields provide the information required for state-valid decision making, but they do not themselves define when a result is accepted or which candidate is selected. Section 4.4 explains how the orchestrator uses this lineage to reject outdated or mismatched evidence and permit only a valid candidate to progress toward bounded actuation.

Field	Carried by	Implemented semantics
`state_id`	`SystemSnapshot`	Identifies the observed physical baseline used to begin a candidate-evaluation cycle.
`based_on_state_id`	`SimulationBatch`, `SimulationBatchReport`	Records the observed state on which the candidate or returned report is based.
`batch_id`	`SimulationBatch`, `SimulationBatchReport`	Correlates a submitted candidate batch with the corresponding simulation report.
`candidate_id`	`Candidate`, `CandidateOutcome`	Correlates an individual candidate with the outcome produced by its simulation.
`created_at`	`SimulationBatchReport`	Records when the report was produced and supports temporal freshness validation.

Table 4.6: Identifiers used to preserve state, candidate, and result lineage.

4.3 Topology-Aware Calibration for Simulation Fidelity

This section answers **RQ2b**: *How can the simulator be calibrated using observed traces so that it maintains sufficient fidelity over time?*

Answer to RQ2b. RQ2b is addressed by estimating the OpenDC CPU power-model calibration factor from recent physical evidence and scoping each estimate to the topology that produced that evidence. A bounded grid search uses mean absolute percentage error (MAPE) over temporally aligned physical and simulated power series to select the estimate, while topology-indexed storage supports exact reuse when a previously calibrated topology reappears in simulation. The mechanism calibrates processor-domain power only and does not imply that fidelity improves uniformly. Chapter 5 further evaluates whether the selected calibration improves the relevant fidelity measures.

The mechanism is scoped to processor-domain power in the prototype. It uses the workload, telemetry, and system snapshot records defined in Section 4.2 as replay input, physical calibration evidence, and configuration context. Within this scope, the implementation realizes **FR5** and supports the power-related part of the fit-for-purpose fidelity required by **NFR1**. It does not calibrate task runtime, processor performance, scheduling, or resource-sharing behavior.

The implementation builds on the window-based calibration method provided by OpenDT (54). This thesis adapts the method to observations produced from live Kubernetes exe-

cution and makes the calibration topology-aware by associating each calibrated parameter value with the worker configuration that produced the physical evidence. The following subsections describe the evidence window, parameter search and error-based selection, and the handling of calibration across topology changes.

4.3.1 Calibration Evidence and Comparison Window

Each calibration sweep combines three forms of evidence with distinct roles. Completed `Task` records provide the workload replay input consumed by OpenDC. `Consumption` records provide timestamped physical processor-domain power as the calibration referent. The latest `TopologySnapshot` provides the physical baseline used for the sweep and identifies the topology with which the selected parameter value is associated. Together, these records provide the replayed workload, physical calibration evidence, and topology context required for the search.

Calibration begins once workload, power, and topology evidence are available over a sufficient interval (30 minutes in the prototype). The implemented calibrator is triggered by evidence readiness instead of by an error threshold. Each subsequent sweep uses newly accumulated evidence without first checking whether the error produced by the current parameter exceeds a fidelity threshold. For each sweep, the simulated and physical power series are first restricted to their common temporal range. The comparator then retains the most recent part of this shared interval, up to a configured maximum (60 minutes).

4.3.2 Bounded Parameter Search and Error-Based Selection

The calibrator estimates the selected simulator parameter through a bounded one-dimensional grid search. For each tested value, it constructs a variant of the currently observed topology by modifying only the selected property. The remaining topology properties and workload input are held constant so that differences in simulated power can be attributed to the tested value. In the current prototype, the calibrated parameter is `cpuPowerModel.calibrationFactor`, which adjusts the OpenDC CPU power model.

Let T denote the observed topology, W the accumulated workload, and Θ the configured parameter grid. For each $\theta \in \Theta$, the calibrator constructs a topology variant T_θ, simulates W under that variant, and compares the resulting processor-power series with the recent physical measurements. All variants therefore evaluate the same workload under otherwise identical topology properties. In the evaluated configuration, Θ contains 15 evenly spaced values from 2 to 9, with at most five simulations executed in parallel.

For a tested value θ, the calibration error is calculated using **mean absolute percentage error (MAPE)**:

$$e(\theta) = \frac{1}{|I_\theta|} \sum_{i \in I_\theta} \left| \frac{P_i^{\text{physical}} - P_{i,\theta}^{\text{simulated}}}{P_i^{\text{physical}}} \right| \times 100\% \tag{4.1}$$

I_θ is the set of aligned observations for which $P_i^{\text{physical}} \neq 0$. P_i^{physical} is the observed processor-domain power, while $P_{i,\theta}^{\text{simulated}}$ is the aligned OpenDC power estimate produced using θ. Physical power forms the denominator because it is the calibration referent. Zero-valued physical observations are excluded to avoid division by zero. Simulations that fail or do not produce a usable aligned comparison are assigned an infinite error, so they cannot outperform a tested value with finite error. A sweep yields a usable calibration result only when at least one tested value produces a finite error.

If at least one tested value produces a finite error, the selected parameter is

$$\theta^* = \arg\min_{\theta \in \Theta} e(\theta) \tag{4.2}$$

It is important to note that this bounded search only identifies the best value within the configured grid. The result depends on the tested interval and spacing, and interactions between multiple uncertain parameters are not considered. A limitation of using MAPE as the calibration objective is that it emphasizes relative pointwise error in power magnitude and does not independently assess temporal delay, task progress, or queueing behavior.

4.3.3 Calibration Across Topology Changes

Topology change introduces a calibration challenge that is absent when a fixed resource configuration is replayed. Candidate generation may change the number of schedulable machines, whereas physical power measurements are available only for the topology currently executing the workload. Evidence from an n-machine execution supports direct calibration of that topology, but not of hypothetical $n-1$- or $n+1$-machine candidates. Using the same physical power series to calibrate every candidate would conflate the effect of topology change with error in the power-model parameter. The prototype therefore calibrates the currently observed topology, which provides the physical baseline for candidate evaluation.

The selected calibration factor is consequently scoped to the topology for which physical evidence is available. The prototype stores calibrated values under a stable topology key that represents the relevant machine configuration. When an exact match is encountered

again during simulation, the stored factor can be used to parameterize that topology rather than reverting immediately to the configured default. If the topology later becomes active in the physical system, a subsequent calibration sweep can refresh the stored value using new physical evidence. The calibration record retains the selected factor, its MAPE, and the evidence-window boundaries, thereby preserving the context in which the value was obtained.

When no exact-topology calibration is available, the prototype uses the configured default power-model value. The broader topology-aware method could additionally reuse a value from the nearest compatible topology, provided that compatibility and topology distance are defined explicitly. Such reuse would constitute an extrapolation from prior evidence instead of direct calibration against the current physical system and should therefore be treated as lower-confidence. Nearest-topology selection is not implemented in the current prototype and is included only as a reference for extending the design.

4.4 From Candidate to Bounded Action

This section answers **RQ2c**: *How can simulator outputs answer what-if questions and be converted into actionable control decisions for the physical system?*

Answer to RQ2c. RQ2c is addressed through a state-bound decision pipeline. The prototype generates a bounded candidate set from the current `SystemSnapshot`, evaluates all candidates under the same accumulated workload, rejects reports that are stale, unknown, or based on a different physical state, and ranks the remaining outcomes under an explicit policy. The selected abstract decision is checked against the current state once more and translated by the controller into a supported Kubernetes node-schedulability operation, after which the physical environment is re-observed. Simulator output therefore serves as decision evidence rather than a direct command. Invalid evidence withholds actuation, while the no-operation candidate allows the policy to retain the current configuration when it ranks that outcome highest.

Figure 4.1 summarizes this state-bound path and the two validation points that guard actuation. The implementation connects a `SystemSnapshot`, a `SimulationBatch`, a `SimulationBatchReport`, an abstract `Decision`, and the controller operation that changes Kubernetes node schedulability. The following subsections describe candidate generation, report validation, policy-based selection, and bounded actuation followed by re-observation. Together, these mechanisms realize **FR3**, **FR4**, **FR6**, and **FR7** and support **NFR2**, **NFR4**, and **NFR5**.

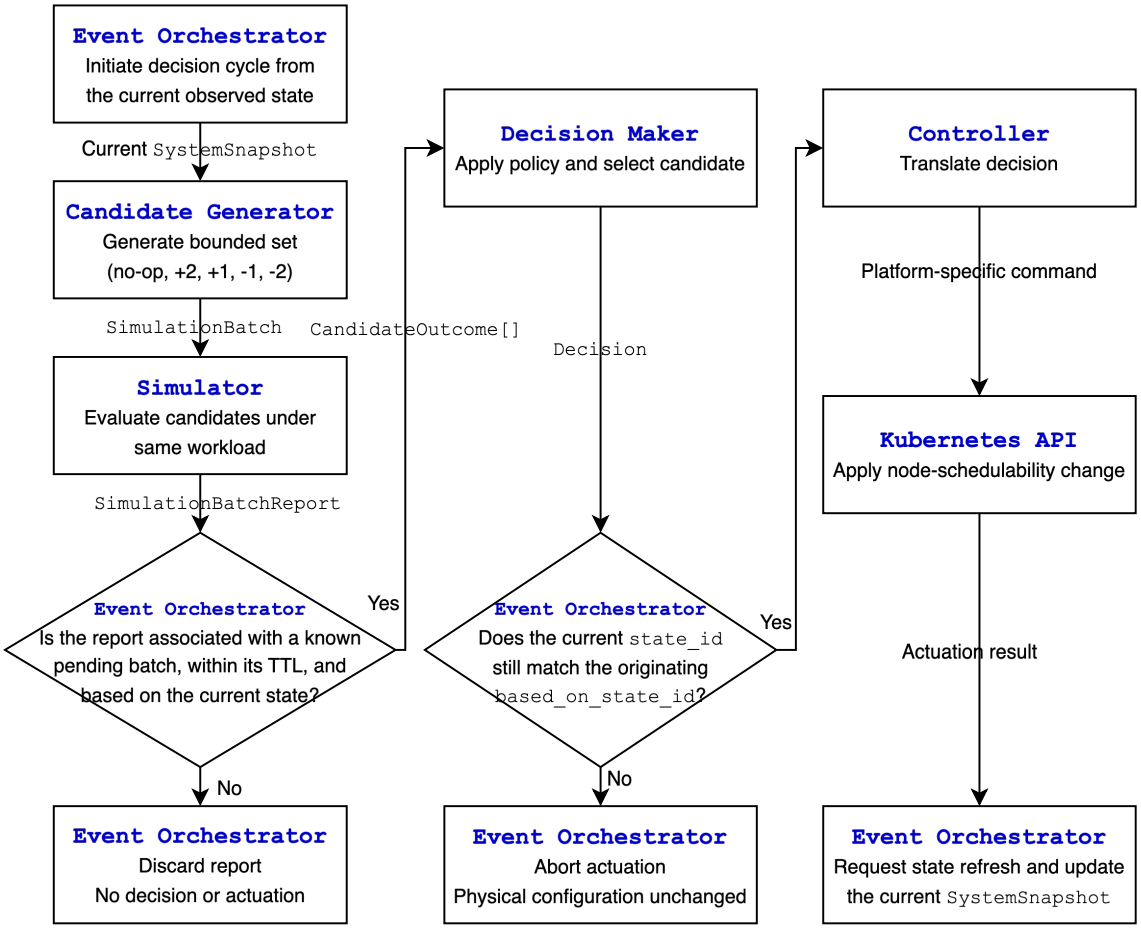

Figure 4.1: State-bound pipeline from candidate generation to bounded Kubernetes actuation.

4.4.1 What-If Candidate Generation

Candidate generation begins from one observed `SystemSnapshot`, which serves as the common physical baseline for every alternative in a decision cycle. The orchestrator assigns this state a `state_id` and requests one `SimulationBatch`, which receives a `batch_id` and `based_on_state_id`. The first candidate is always a no-operation candidate, which retains the observed topology and carries a `no_op` decision, thereby representing the configuration already in use. The remaining candidates modify this baseline. When the simulator evaluates the batch, all candidates use the same accumulated trace-derived workload and the same analysis interval. Differences between their outcomes can therefore be interpreted as consequences of their candidate topologies rather than differences in input workload.

Each `Candidate` contains two complementary fields. The `decision` field expresses the platform-independent action that may later be selected, while the `candidate_topology`

field represents the resulting configuration in the form required by OpenDC. For a node-count alternative, the decision uses the `change_node_count` action and records the node counts before and after the proposed change. The corresponding `candidate_topology` represents the same target configuration using the simulator-facing topology representation introduced in Section 4.2.3. For the no-operation candidate, the decision specifies no change, and the candidate topology remains equal to the observed baseline.

The evaluated prototype restricts this decision space to fixed alternatives for the number of schedulable cloud workers. Relative alternatives request changes of $+2$, $+1$, -1, and -2 workers. Target counts are bounded between one and the available capacity recorded in the current `SystemSnapshot`. A request that produces the current topology is omitted because it would duplicate the no-operation candidate. These alternatives define the experimental decision space of the prototype rather than a general limitation of the architecture. Other candidate generators could represent placement, resource-limit, processor-frequency, or admission-control decisions while preserving the same candidate contract.

4.4.2 Simulation-Report Validation

Candidate evaluation proceeds asynchronously, so a returned `SimulationBatchReport` is not immediately eligible for policy selection. The topology or available-capacity bound represented by the originating state may have changed while the simulator was evaluating the batch. The orchestrator therefore validates the report before examining which candidate appears preferable. These checks use the lineage fields introduced under the operational trace contract. They establish whether the report remains temporally relevant and whether it assumes the same resource baseline as the most recently observed state.

A report returned by the simulator is discarded by the orchestrator when any of the following conditions holds:

- No current observed state is available.
- The report's `based_on_state_id` differs from the current `state_id`.
- The report's `batch_id` does not identify a known pending batch.
- The report's age, measured from `created_at`, exceeds the configured report time-to-live.

The simulator assigns `created_at` when it constructs the report after completion of candidate evaluation. The time-to-live bounds asynchronous transport, queueing, and event-processing delay between report production and decision selection. The discard conditions prevent old, state-mismatched, or unrecognized evidence from entering policy selection. Discarding a report does not create an explicit no-operation decision. It withholds actuation and leaves the physical configuration unchanged.

Some reports are deferred rather than discarded. The orchestrator requeues a report when a state refresh is being processed. Deferral gives the higher-priority physical-state operation time to complete before the report is reconsidered. The report is then subjected to the same age and state checks again, so a deferred report may still be discarded if it has become too old or its originating state has been replaced.

4.4.3 Policy-Based Selection of an Abstract Decision

After report validation, the `CandidateOutcome`s are extracted from `SimulationBatchReport`. Each outcome contains a set of named numeric metrics representing the simulator's predicted behavior for that candidate. These values provide the basis for comparing candidates without requiring the decision policy to inspect the simulator's output files. Each metric contains a numeric value and an optional unit. The current implementation defines the following outcome metrics:

- `runtime`: the span between the earliest and latest service timestamps, reported in seconds;
- `utilization`: the mean processor-utilization rate over the analyzed host samples, represented as a dimensionless value;
- `power`: the mean simulated processor-power draw over the analyzed interval, reported in watts.

The prototype converts these metrics into an ordering of the candidates through either a weighted or ranked policy. Each objective specifies the metric to consider and whether the policy should minimize or maximize it. A weighted policy normalizes the selected metrics across the candidates in the current report before combining them according to their configured weights. Normalization allows metrics with different units and numeric ranges, such as runtime and power, to contribute to one aggregate score. Metrics unavailable for an individual candidate do not contribute to its score, while a candidate without any usable objective receives the lowest score.

A ranked policy instead compares objectives lexicographically according to their configured priorities. Candidates are first ordered using the highest-priority objective. Candidates whose values differ by no more than the configured tolerance are treated as tied and compared using the next objective. Missing values are ordered after available values for the corresponding objective. This policy can therefore express runtime as the primary objective and power as a tie-breaker, or reverse the ordering for a power-oriented configuration, without changing candidate generation or controller implementation.

The decision maker returns the abstract `Decision` attached to the highest-ranked candidate. The no-operation candidate participates in the same comparison as every node-count alternative, allowing the policy to retain the current physical configuration when its predicted outcome is preferable.

The implemented policies compare the reported metric values directly after the orchestrator has completed the state-validity checks. They do not require a minimum predicted improvement over the no-operation candidate and do not incorporate calibration error or prediction uncertainty into candidate ranking. These mechanisms remain extensions rather than part of the implemented decision policy. Consequently, the prototype only partially realizes the fidelity-aware selection aspect of **NFR1**.

4.4.4 Bounded Controller Actuation and Re-observation

The controller is the only component that translates a selected abstract decision into a physical-system operation and applies it to the execution environment. In the prototype, it supports the `no_op` and `change_node_count` action kinds specified by the decision. A no-operation decision completes without modifying Kubernetes. For a node-count decision, the controller reads the target counts from the decision details and translates them into Kubernetes API operations. Candidate generation, simulation, and policy selection consequently remain independent of the platform-specific mechanism used to enact the selected change. Immediately before invoking the controller, the orchestrator repeats the state-validity check and aborts actuation if the current `state_id` no longer matches the `based_on_state_id` of the validated report.

Actuation is recorded as an observable operation. For each selected decision, the data store records the state identifier against which the decision was applied, the decision contents, and whether the controller operation completed successfully. A timeout or exception is recorded together with an error message.

After the actuation attempt, the orchestrator observes the physical environment again. If the refreshed topology or available-capacity boundary differs from the preceding snapshot,

the new observation receives a new `state_id`, and a new decision cycle begins from that state.

5

Evaluation

This chapter presents the experimental evaluation of the trace-based closed-loop digital twin realized in Chapter 4. It first describes the experimental platform, workload, experimental settings, metrics, and analysis procedure shared across the evaluation (Section 5.1). It then conducts three evaluation experiments: **(1)** Assessing whether OpenDC provides fit-for-purpose fidelity under fixed worker topologies by comparing task progress, timing, processor-power magnitude, and processor-power trend (Section 5.2). **(2)** Evaluating whether topology-aware calibration improves processor-power fidelity across the static configurations, the pending-Pod baseline, and the runtime-oriented and power-oriented closed-loop policies (Section 5.3). **(3)** Answering **RQ3** by comparing the physical workload runtime and RAPL-derived processor-domain energy of the two closed-loop policies with the pending-Pod baseline, while using the static settings as reference points (Section 5.4). Finally, the chapter discusses threats to validity (Section 5.5).

5.1 Experimental Setup

All experiments use the same physical testbed, generated workload, and observation pipeline. This section describes the experimental platform, workload, experimental settings, and analysis procedure shared by the subsequent experiments.

5.1.1 Experimental Platform

The experiments use the prototype environment introduced in Chapter 4. This subsection reports the deployment and measurement settings that affect the evaluation. Continuum deploys nine QEMU/KVM virtual machines across five physical hosts. One virtual machine serves as the Kubernetes control plane, while the remaining eight form the worker pool

Property	Configuration
Number of physical hosts	5
Physical processor	Intel Xeon Silver 4210R @ 2.40 GHz
Host operating systems	Ubuntu 22.04.3 on `node1`, Ubuntu 22.04.1 on `node2`, Ubuntu 22.04.5 on `node3`, and Ubuntu 20.04.6 on `node4` and `node5`
Virtualization	QEMU/KVM through Continuum
Kubernetes topology	One control-plane VM and eight worker VMs
Resources per VM	8 virtual CPU cores and 32 GB of memory
CPU configuration	CPU pinning enabled
Worker placement	Two worker VMs on each of four physical worker hosts
Guest operating system	Ubuntu 20.04.6
Kubernetes version	`v1.27.16`
OpenDC version	`v2.4f`

Table 5.1: Physical, virtual, and software configuration of the experimental platform.

whose schedulability is varied during the experiments. Each virtual machine is allocated 8 virtual CPU cores with CPU pinning enabled and 32 GB of memory.

The eight worker virtual machines are distributed evenly across four physical hosts (from `node1` through `node4`), with two workers placed on each host. The fifth physical host (`node5`) runs the closed-loop prototype services and hosts the Kubernetes control-plane virtual machine. Prometheus is constrained to the control-plane node so that its monitoring workload does not compete for resources with the experimental Jobs on the workers.

Table 5.1 summarizes the physical hardware, virtual-cluster allocation, and software versions used across all experiments. Prometheus and Scaphandre collect resource-use and RAPL-derived energy measurements at 15-second intervals. A separate Scaphandre source is configured for each worker virtual machine, and the resulting measurements are aggregated across the eight-worker pool. All eight worker virtual machines remain provisioned and monitored even when some workers are marked as unschedulable. As established in Chapter 4, these measurements cover processor-related RAPL domains and should not be interpreted as total machine or cluster power.

5.1.2 Workload

The evaluation uses a synthetic CPU-bound batch workload with a controlled change in arrival intensity. The workload contains a total of 370 finite Kubernetes Jobs and follows a low–medium–high–medium–low arrival profile. This profile exposes the physical system

Stage	Arrival intensity	Number of Jobs	Mean inter-arrival time μ_s
1	Low	25	28 seconds
2	Medium	80	20 seconds
3	High	160	10.5 seconds
4	Medium	80	20 seconds
5	Low	25	28 seconds
Total		370	–

Table 5.2: Staged arrival configuration of the evaluation workload.

and simulator to both increasing and decreasing demand rather than evaluating them under a single constant arrival rate.

The Poisson arrival model introduces stochastic variation around a controlled mean inter-arrival time and avoids an artificial deterministic submission schedule. Varying this mean between arrival stages produces a repeatable change in offered load, allowing the evaluation to examine whether the physical system and simulator follow changes in workload intensity over time. Therefore, the workload is a controlled approximation of irregular Job arrivals instead of a reproduction of a specific production trace.

Each Job executes a CPU-burn container and specifies requests and limits one CPU core and 512 MiB of memory. A Job begins with a 60-second CPU-burn warm-up, followed by a steady CPU-burn interval whose duration is sampled uniformly from the range of 300 to 600 seconds. It then remains active for a 60-second cooldown interval before completion. The Jobs are independent and contain no inter-task dependencies.

Table 5.2 presents the number of Jobs and mean inter-arrival time used in each stage. Within stage s, the inter-arrival time X_s is sampled independently from an exponential distribution,

$$X_s \sim \mathrm{Exp}(\lambda_s), \qquad \lambda_s = \frac{1}{\mu_s} \tag{5.1}$$

where μ_s is the configured mean inter-arrival time for that stage.

A fixed seed controls the sampled inter-arrival times, steady-burn durations, and initial state of the CPU-burn program. Thus, each experimental setting receives the same sampled arrival and duration sequences.

5.1.3 Experimental Settings

The evaluation combines five resource-management strategies with calibration disabled or enabled, producing the ten experimental settings **(ES)** summarized in Table 5.3. The

Strategy	Calibration disabled	Calibration enabled	Control basis
Eight-worker static	ES1	ES6	Fixed worker count
Four-worker static	ES2	ES7	Fixed worker count
Pending-Pod baseline	ES3	ES8	Pending-Pod count
Closed-loop runtime-oriented policy	ES4	ES9	Minimum predicted runtime
Closed-loop power-oriented policy	ES5	ES10	Minimum mean simulated processor power

Table 5.3: Experimental settings and corresponding resource-management strategies.

settings are identified as ES1–ES10 throughout the remainder of this chapter. Each setting is executed three times, resulting in a total of 30 physical executions. The repetitions aim to capture variation in physical execution, scheduling, and observation while holding the generated workload and setting definition constant.

The first two strategies keep a fixed number of workers schedulable throughout the workload execution. The pending-Pod baseline is implemented as an adaptation of Kubernetes Node Autoscaling (67). It follows the same reactive principle, in which unschedulable Pods indicate that additional node capacity is required and surplus capacity may subsequently be reduced. In this adaptation, scale-out and scale-in are emulated by changing worker schedulability within the fixed pool of eight existing workers instead of provisioning or removing machines.

The two closed-loop strategies use the same candidate space and bounded node-schedulability actions but rank the simulated candidates under different objectives. The runtime-oriented policy selects the candidate with the lowest predicted workload runtime. The power-oriented policy selects the candidate with the lowest mean simulated processor power.

The calibrated settings use the topology-aware calibration mechanism described in Section 4.3. The workload, resource-management strategy, initial worker count, and worker-capacity bound remain unchanged between each uncalibrated setting and its calibrated counterpart.

All settings except ES1 and ES6 begin with four schedulable workers. The pending-Pod baseline polls the workload queue every 210 seconds. When at least ten Pods are pending, it makes one additional existing worker schedulable. When the pending-Pod count has remained zero for at least 60 seconds, the baseline marks one worker as unschedulable at the next polling step.

The closed-loop settings refresh the observed system state every 30 seconds and initiate periodic candidate evaluation every five minutes. To avoid excessively long experimental runs when simulator-informed decisions retain too few schedulable workers, these settings also include a backlog-relief rule. Every 210 seconds, the decision maker receives the pending-Pod count from the observer. When at least ten Pods are pending, the decision maker produces a decision to make one additional existing worker schedulable. The backlog-relief rule operates independently of simulator-informed candidate ranking.

5.1.4 Metrics and Analysis Procedures

This subsection defines the metrics and common analysis procedures used to assess task progress and timing, processor-power fidelity, the effect of calibration, and the physical outcomes of closed-loop control.

Task progress and timing. All task timestamps are converted to elapsed time relative to the earliest Job submission in the corresponding physical or simulated execution. Workload makespan is the interval between this first submission and the final Job completion. The **signed relative makespan error** is

$$\epsilon_M = \frac{M^{\mathrm{sim}} - M^{\mathrm{phys}}}{M^{\mathrm{phys}}} \tag{5.2}$$

where M^{phys} and M^{sim} denote the physical and simulated makespans, respectively. A positive value indicates that the simulation completes later than the physical execution, while a negative value indicates that it completes earlier.

Makespan alone does not show whether the physical and simulated workloads progress similarly during execution. A cumulative-completion curve $C(t)$ is used to record the number of completed Jobs at elapsed time t. To quantify the curve similarity, physical and simulated completion counts are first sampled on a shared 60-second grid that extends until both executions have completed. Their difference is then summarized using the **cumulative-completion normalized root mean squared error (NRMSE)**,

$$\mathrm{NRMSE} = \frac{1}{N}\sqrt{\frac{1}{m}\sum_{j=1}^{m}\left(C^{\mathrm{sim}}(t_j) - C^{\mathrm{phys}}(t_j)\right)^2} \tag{5.3}$$

where m is the number of aligned time points and $N = 370$ is the total number of Jobs. The normalization expresses the error relative to the workload size.

Wait time and **turnaround time** are examined as supporting task-level diagnostics. They are defined as

$$T_{\text{wait}} = t_{\text{start}} - t_{\text{submission}}, \qquad T_{\text{turnaround}} = t_{\text{finish}} - t_{\text{submission}} \tag{5.4}$$

The distributions are summarized using their medians and 95th percentiles. These diagnostics help identify differences in queueing and end-to-end Job delay.

Processor-power preprocessing. Observed processor-domain power and simulated processor power are compared only over their common temporal range. For each execution, the two series are resampled and interpolated onto a shared 60-second grid, after which a centered 15-minute moving average is applied to both series. This preprocessing reduces sensitivity to individual samples and small sampling-time differences while retaining workload-scale changes. All processor-power fidelity metrics are calculated from the resulting smoothed series.

Processor-power magnitude. Let P_i^{obs} denote observed processor-domain power and P_i^{sim} denote simulated processor power at aligned time point i. Power-magnitude error is assessed using **mean absolute error (MAE)** and **symmetric mean absolute percentage error (sMAPE)**,

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}\left|P_i^{\text{sim}} - P_i^{\text{obs}}\right| \tag{5.5}$$

$$\text{sMAPE} = \frac{1}{n}\sum_{i=1}^{n}\frac{2\left|P_i^{\text{sim}} - P_i^{\text{obs}}\right|}{\left|P_i^{\text{obs}}\right| + \left|P_i^{\text{sim}}\right|} \times 100\% \tag{5.6}$$

MAE reports the average watt-level difference between the two series, while sMAPE expresses the pointwise difference relative to the magnitudes of both values, which supports comparison across settings with different processor-power ranges. For both metrics, lower values indicate greater similarity between the physical and simulated power magnitudes.

Globally aligned processor-power trend. Power-trend fidelity is assessed separately from watt-level magnitude. Each complete smoothed series is standardized independently,

$$Z_i^x = \frac{P_i^x - \overline{P^x}}{\sigma_{P^x}} \qquad x \in \{\text{obs}, \text{sim}\}, \tag{5.7}$$

where $\overline{P^x}$ and σ_{P^x} are the mean and standard deviation of series x. The **globally aligned normalized-shape error** is then

$$\text{zRMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(Z_i^{\text{sim}} - Z_i^{\text{obs}}\right)^2} \tag{5.8}$$

Lower zRMSE indicates greater similarity between the normalized curves while preserving their original temporal alignment. Since each series is standardized independently, zRMSE does not assess similarity in absolute watt magnitude.

Lag-tolerant local processor-power trend. Inspired by Xie's segmented correlation assessment for power-trend fidelity (68), this evaluation uses a **fixed-duration segmented best-lag Pearson correlation** as a supporting trend metric. The adopted procedure retains Xie's bounded per-segment lag search and mean aggregation, but the segment and lag parameters are adjusted to suit the present data.

The standardized series are divided into consecutive, non-overlapping 30-minute segments. Each complete segment contains 30 observations on the 60-second grid. A trailing segment shorter than 30 minutes is excluded so that all evaluated segments represent the same duration. This segmentation is independent of the workload arrival stages and is not described as a phase-based analysis.

Within each complete segment, Pearson correlation is calculated over temporal lags from -10 to $+10$ minutes in 60-second increments. Only a lag that retains at least 20 paired observations is considered. A segment is excluded as low-variation when the standard deviation of either standardized segment is below 0.1.

Let $\rho_k(\ell)$ denote the Pearson correlation for segment k after applying a valid lag ℓ. The best correlation for that segment is

$$r_k^{\star} = \max_{\ell \in \mathcal{L}_k} \rho_k(\ell) \tag{5.9}$$

where $\mathcal{L}_k$ contains the searched lags that satisfy the minimum-pair requirement. The segmented correlation r_{seg} is the mean across the K valid segments,

$$r_{\text{seg}} = \frac{1}{K}\sum_{k=1}^{K} r_k^{\star} \tag{5.10}$$

Higher r_{seg} indicates stronger local similarity in the rises and falls of the processor-power series after allowing a bounded temporal displacement.

The segmented correlation is interpreted as supporting evidence rather than as an independent demonstration of fidelity. Selecting a separate lag within each segment can

Dimension	Metric	Interpretation
Task progress	Signed relative makespan error and cumulative-completion NRMSE	Agreement in total workload duration and completion progress
Task timing	Wait-time and turnaround-time distributions	Diagnostic evidence of queueing and end-to-end Job delay
Power magnitude	MAE and sMAPE	Absolute watt-level error and relative power-magnitude error
Power trend	zRMSE and fixed-duration segmented best-lag Pearson correlation	Globally aligned normalized-shape error and lag-tolerant local trend agreement
Calibration	Change in the four processor-power fidelity metrics	Whether calibration improves magnitude or trend fidelity
Physical outcomes	Workload runtime and RAPL-derived processor-domain energy	Runtime-energy outcome relative to the calibration-disabled pending-Pod baseline ES3

Table 5.4: Summary of the metrics used in the evaluation.

overstate agreement when the required displacement changes during an execution. Therefore, it is interpreted together with zRMSE, which retains the original global alignment, and with MAE and sMAPE, which assess power magnitude.

Calibration analysis. Calibration is evaluated using the same four processor-power fidelity metrics: MAE, sMAPE, zRMSE, and r_{seg}. A reduction in MAE, sMAPE, or zRMSE indicates improvement in the corresponding error dimension, while an increase in r_{seg} indicates stronger lag-tolerant local trend agreement. The metrics are considered separately. This permits calibration to improve one aspect of fidelity while worsening another.

Physical outcomes and runtime–energy trade-off. For the decision-support analysis, **physical workload runtime** is measured from the earliest Job submission to the latest Job completion. **RAPL-derived processor-domain energy** is calculated over the same workload interval by summing the recorded interval energy increments across all eight worker virtual machines. All eight workers are included because an unschedulable worker remains provisioned and monitored. The resulting quantity covers processor-related RAPL domains and is not interpreted as total machine, cluster, or testbed energy.

Table 5.4 summarizes the metric dimensions, their corresponding measures, and their interpretation.

5.1.5 Aggregation and Variability Reporting

Results aggregated across the three repetitions of each experimental setting are reported as the arithmetic mean ± sample standard deviation. In figures, lines and bars show the arithmetic mean, while shaded bands and error bars show the mean ± one sample standard deviation across the three repetitions.

5.2 Fit-for-Purpose Simulation Fidelity under Static Topologies

This section provides the validation experiment that assesses whether OpenDC achieves fit-for-purpose fidelity for the workload characteristics used by the closed-loop prototype under fixed worker topologies. The experiment compares ES1, the uncalibrated eight-worker static setting, with ES2, the uncalibrated four-worker static setting. Calibration is disabled in both settings to isolate baseline simulator fidelity from the topology-aware calibration evaluated in Section 5.3.

The results show that fidelity depends on the topology. In ES1, OpenDC reproduces cumulative Job progress and total makespan closely and retains strong processor-power trend agreement. In ES2, it completes the workload too early, misses the long physical completion tail, and reproduces the globally aligned power trend less accurately. Power-magnitude errors remain on a similar scale in both settings. In short, OpenDC is more faithful under the eight-worker topology, while the four-worker topology exposes limitations in its representation of queueing and power behavior under resource constraints.

5.2.1 Task Progress and Timing

This subsection evaluates whether OpenDC reproduces workload progress and Job timing under the two static topologies. OpenDC closely reproduces both in ES1, but it does not reproduce the timing tail observed in ES2. Figure 5.1 compares cumulative Job completions in the physical and simulated executions, while Table 5.5 reports the physical and simulated timing diagnostics.

For ES1, the two completion curves are nearly indistinguishable throughout the workload. The signed makespan error is $-0.083\% \pm 0.046\%$, and the cumulative-completion NRMSE is $0.215\% \pm 0.005\%$. The timing diagnostics in Table 5.5 also show close agreement in wait time and turnaround time under this setting. Although OpenDC omits the short

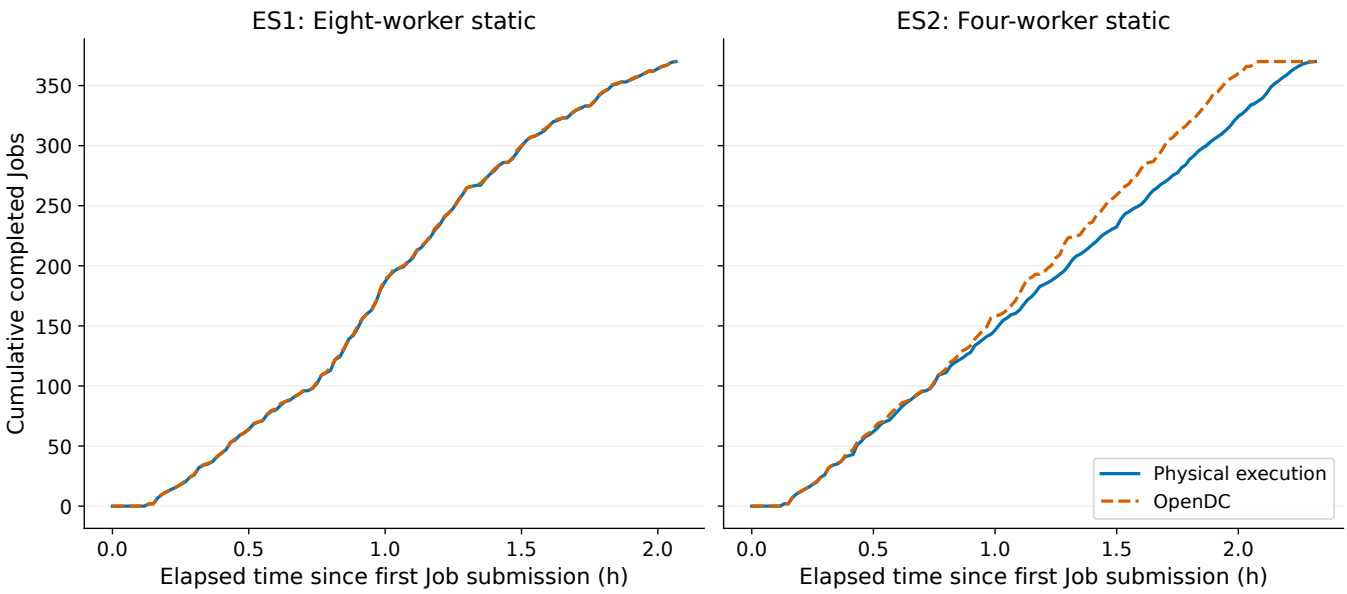

Figure 5.1: Cumulative Job completion for physical execution and OpenDC under ES1 and ES2.

Metric	ES1 physical	ES1 OpenDC	ES2 physical	ES2 OpenDC
Makespan (s)	7380.33 ± 4.62	7374.24 ± 4.46	8253.33 ± 60.21	7456.82 ± 5.41
Median wait time (s)	1.0 ± 0.0	0.0 ± 0.0	68.0 ± 5.1	300.8 ± 2.1
95th-percentile wait time (s)	5.0 ± 0.0	0.0 ± 0.0	3293.6 ± 338.2	712.9 ± 2.5
Median turnaround time (s)	557.5 ± 1.3	549.6 ± 2.7	676.7 ± 4.5	860.9 ± 2.9
95th-percentile turnaround time (s)	707.7 ± 0.6	698.9 ± 0.4	3919.4 ± 296.2	1345.0 ± 2.0

Table 5.5: Task-timing diagnostics for physical execution and OpenDC under ES1 and ES2.

platform-side wait time observed physically, this difference has little effect on the overall completion trajectory.

For ES2, the simulated curve moves ahead of the physical curve during the latter half of the execution and reaches all 370 completions earlier. The signed makespan error is $-9.648\% \pm 0.667\%$, while the cumulative-completion NRMSE increases to $5.083\% \pm 0.203\%$. The timing diagnostics show that this mismatch is not a uniform offset. OpenDC overestimates the medians of the wait-time and turnaround-time distributions but substantially underestimates their 95th percentiles. OpenDC redistributes predicted delay across Jobs without reproducing the extreme queueing that forms the physical completion tail.

The task-progress results show that OpenDC reproduces execution closely when eight workers provide sufficient parallel capacity. Under the four-worker constraint, the simulator does not reproduce the tail behavior of the physical execution. Simulated timing should consequently be treated as comparative evidence rather than an exact prediction

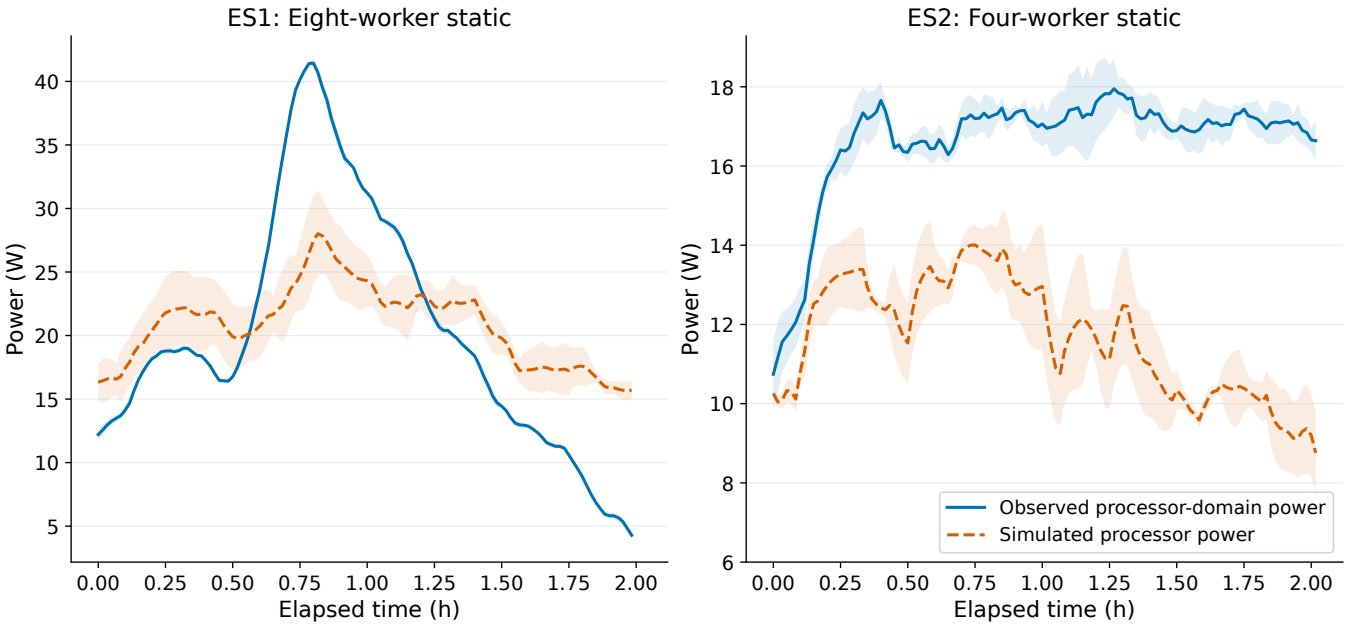

Figure 5.2: Observed processor-domain power and simulated processor power under ES1 and ES2.

for similarly constrained workloads.

5.2.2 Processor-Power Magnitude and Trend

This subsection evaluates whether OpenDC reproduces the magnitude and temporal trend of observed processor-domain power. Watt-level error is moderate in both settings, but trend fidelity is clearly stronger in ES1.

Figure 5.2 compares the power magnitudes. In ES1, OpenDC follows the broad rise and decline of observed processor-domain power, but it compresses the main peak and remains above the observed series during the final part of the common interval. In ES2, observed power remains within a relatively narrow range after its initial rise, while simulated power generally declines during the latter half of the interval. Table 5.6 further shows that ES2 has a slightly lower MAE but a higher sMAPE than ES1. Its lower absolute error does not indicate stronger magnitude fidelity because the observed power range is also lower.

Figure 5.3 compares the standardized trends. ES1 captures the principal rise, peak, and decline of processor power. ES2 diverges from the observed trajectory over extended parts of the run. The zRMSE increases from 0.453 in ES1 to 1.285 in ES2, while the segmented best-lag correlation for ES1 is 0.874 and that of ES2 is 0.660. Both trend metrics indicate stronger processor-power trend fidelity in ES1 than in ES2.

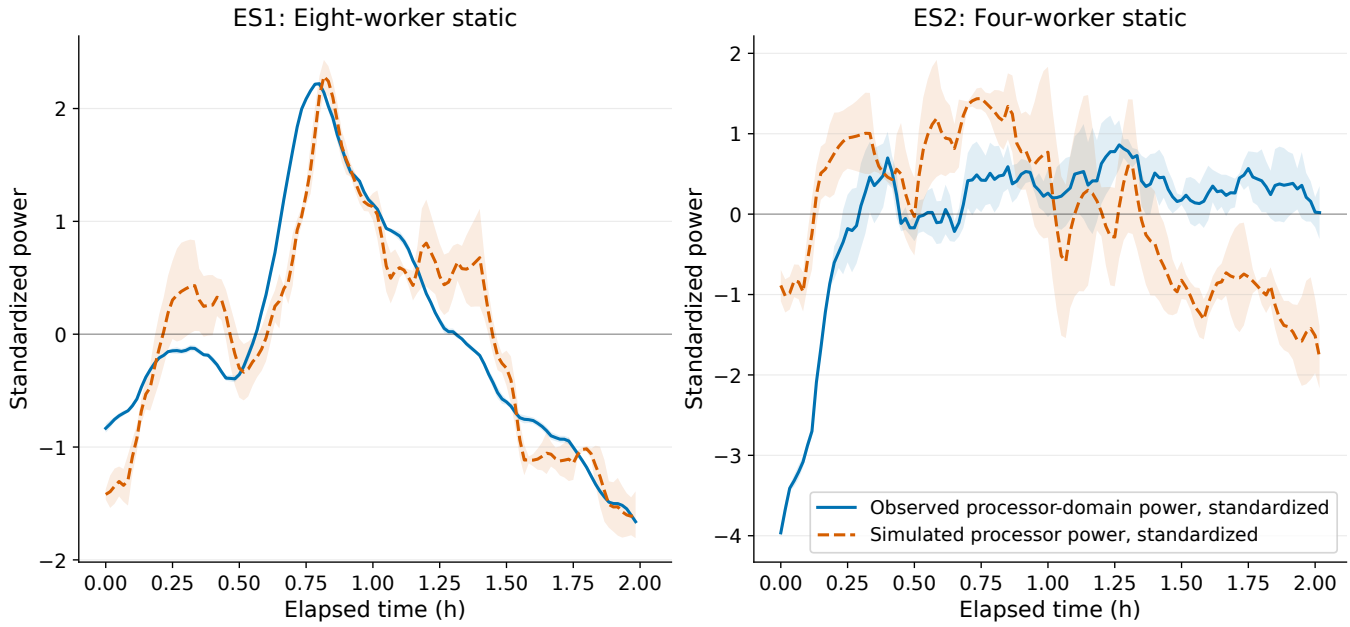

Figure 5.3: Standardized trends of observed processor-domain power and simulated processor power under ES1 and ES2.

Metric	ES1	ES2
Signed makespan error (%)	-0.083 ± 0.046	-9.648 ± 0.667
Cumulative-completion NRMSE (%)	0.215 ± 0.005	5.083 ± 0.203
Processor-power MAE (W)	5.876 ± 0.400	4.970 ± 0.163
Processor-power sMAPE (%)	31.393 ± 2.236	35.430 ± 1.462
Processor-power zRMSE	0.453 ± 0.027	1.285 ± 0.006
Segmented best-lag r_{seg}	0.874 ± 0.036	0.660 ± 0.112

Table 5.6: Summary of simulator fidelity under the static topologies ES1 and ES2.

Table 5.6 summarizes the principal task-progress and processor-power metrics. In brief, the static-topology results provide conditional evidence of fit-for-purpose fidelity. ES1 shows near-zero makespan error, low cumulative-completion error, and stronger globally aligned power-trend agreement, while ES2 does not reproduce the physical completion tail or the global power trend with comparable fidelity. These limitations motivate evaluating calibration separately and treating simulation outcomes as comparative evidence rather than exact physical predictions.

5.3 Effect of Topology-Aware Calibration on Processor-Power Fidelity

This section evaluates whether the topology-aware calibration introduced in Section 4.3 improves processor-power fidelity. It compares each experimental setting with calibration disabled against the corresponding setting with calibration enabled: ES1 against ES6 for the eight-worker static topology, ES2 against ES7 for the four-worker static topology, ES3 against ES8 for the pending-Pod baseline, ES4 against ES9 for the runtime-oriented policy, and ES5 against ES10 for the power-oriented policy.

For MAE, sMAPE, and zRMSE, lower values indicate better fidelity, while a higher segmented best-lag correlation r_{seg} indicates stronger local trend agreement. The main result is that the implemented topology-aware calibration does not consistently improve processor-power fidelity. Across the five paired settings, no calibrated setting improves all four fidelity metrics, and the direction and magnitude of the effect vary across metrics and experimental settings. Overall, the effect of the implemented calibration is metric- and setting-dependent rather than uniformly beneficial. Topology-aware scoping preserves the configuration context of the calibration evidence, but the resulting parameter adjustment does not guarantee improved processor-power fidelity across the evaluated metrics.

Figure 5.4 shows that mean sMAPE increases after calibration in every pair. The increase is small for the runtime-oriented pair and largest for the pending-Pod baseline, which also exhibits greater between-run variation when calibration is enabled. Since lower sMAPE indicates better relative magnitude fidelity, these results do not indicate an improvement in relative power-magnitude accuracy.

Figure 5.5 shows a different result for the globally aligned power trend. Calibration reduces mean zRMSE for the ES1–ES6, ES3–ES8, and ES4–ES9 pairs. The largest reduction occurs for the pending-Pod baseline, while the largest increase occurs for the power-oriented pair. Calibration consequently reduces globally aligned trend error in three pairs but degrades it in two.

Table 5.7 provides the complete four-metric comparison. In the ES1–ES6 pair, both magnitude metrics worsen while both trend metrics improve. All four metrics worsen in the ES2–ES7 pair. The ES3–ES8 pair improves in MAE and zRMSE but worsens in sMAPE and segmented correlation. The ES4–ES9 pair improves in both trend metrics, while magnitude error is approximately unchanged or slightly worse. All four metrics worsen in the ES5–ES10 pair. Interpreting each metric according to its intended direction

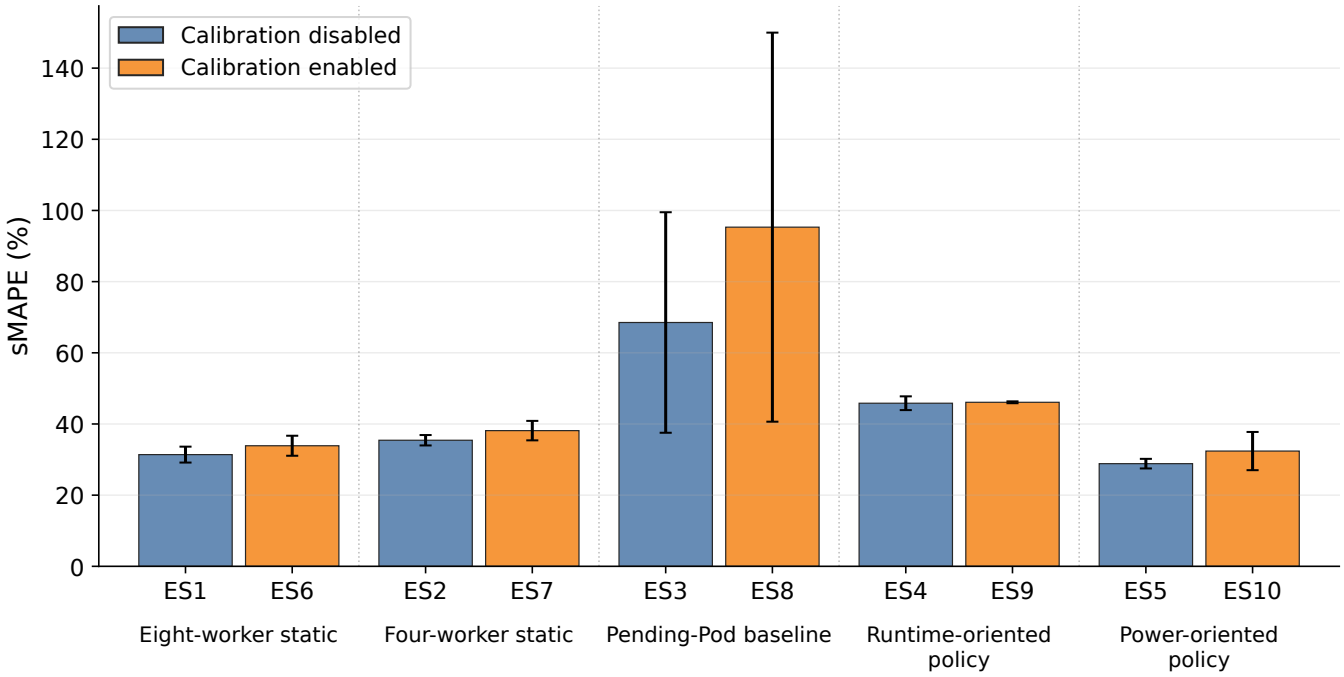

Figure 5.4: Processor-power sMAPE with calibration disabled and enabled. Lower values indicate better relative magnitude fidelity.

thus shows that the current calibration can improve one aspect of fidelity while degrading another, and it is not uniformly beneficial.

The mixed outcome follows from the difference between the implemented calibration objective and the complete-run evaluation. The calibrator selects one CPU power-model calibration factor by minimizing MAPE over a recent evidence window, whereas this evaluation assesses complete executions using MAE, sMAPE, zRMSE, and r_{seg}. A factor that improves windowed MAPE can therefore still worsen another magnitude or trend measure over the full execution. In addition, a single power-model factor cannot correct discrepancies caused by execution timing, scheduling, queueing, or resource sharing.

Topology-aware association still has value because it prevents physical evidence from one worker topology from being treated as direct calibration evidence for another. However, topology scoping addresses the validity context of the evidence, not every source of simulator error. A broader calibration design could search multiple parameters or optimize a defined set of decision-critical metrics. These alternatives are not implemented or evaluated in this prototype and remain future work.

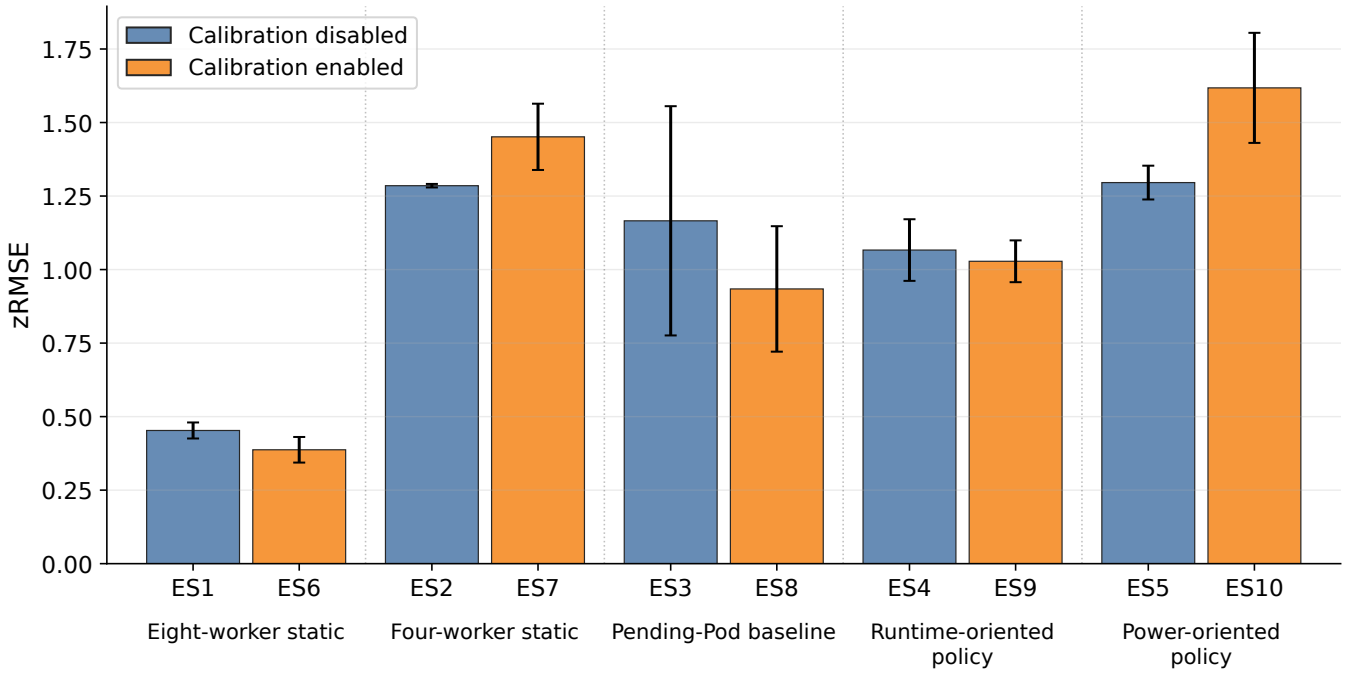

Figure 5.5: Processor-power zRMSE with calibration disabled and enabled. Lower values indicate better globally aligned trend fidelity.

Strategy	Experimental setting	MAE (W)	sMAPE (%)	zRMSE	r_{seg}
Eight-worker static	ES1, calibration disabled	5.876 ± 0.400	31.393 ± 2.236	0.453 ± 0.027	0.874 ± 0.036
	ES6, calibration enabled	6.217 ± 0.401	33.883 ± 2.813	0.387 ± 0.044	0.917 ± 0.113
Four-worker static	ES2, calibration disabled	4.970 ± 0.163	35.430 ± 1.462	1.285 ± 0.006	0.660 ± 0.112
	ES7, calibration enabled	5.270 ± 0.342	38.134 ± 2.735	1.451 ± 0.113	0.492 ± 0.165
Pending-Pod baseline	ES3, calibration disabled	9.070 ± 2.281	68.518 ± 30.987	1.166 ± 0.390	0.860 ± 0.052
	ES8, calibration enabled	8.628 ± 1.930	95.309 ± 54.665	0.934 ± 0.213	0.654 ± 0.059
Runtime-oriented policy	ES4, calibration disabled	7.064 ± 0.419	45.835 ± 1.932	1.066 ± 0.105	0.624 ± 0.203
	ES9, calibration enabled	7.259 ± 0.012	46.095 ± 0.261	1.028 ± 0.071	0.796 ± 0.238
Power-oriented policy	ES5, calibration disabled	3.389 ± 0.098	28.850 ± 1.350	1.296 ± 0.057	0.714 ± 0.099
	ES10, calibration enabled	3.786 ± 0.680	32.389 ± 5.370	1.618 ± 0.187	0.501 ± 0.297

Table 5.7: Processor-power fidelity with calibration disabled and enabled. Lower is better for MAE, sMAPE, and zRMSE; higher is better for r_{seg}.

5.4 Decision-Support Value of Closed-Loop Control

This section evaluates **RQ3** by comparing the physical workload runtime and RAPL-derived processor-domain energy of the runtime-oriented and power-oriented closed-loop policies with the pending-Pod baseline. The comparison uses the calibration-disabled settings only: ES4 and ES5 are compared with ES3, while ES1 and ES2 provide fixed-topology reference points. We exclude the calibrated counterparts from the primary **RQ3** comparison because the implemented calibration adjusts only a CPU power-model parameter and does not calibrate the physical runtime-energy outcomes evaluated here.

The results show that the runtime-oriented policy provides the stronger practical trade-off. Relative to ES3, ES4 keeps runtime nearly unchanged while reducing processor-domain energy. ES5 reduces energy only slightly further, but increases runtime by approximately one-third. The closed loop therefore provides qualified decision-support value under the evaluated workload, but the value depends on the objective used to rank candidates.

5.4.1 Physical Runtime and Processor-Domain Energy

Figures 5.6 and 5.7 report the physical outcomes for ES1–ES5. Table 5.8 provides the corresponding numerical results. Percentage changes for ES4 and ES5 are calculated relative to the pending-Pod baseline ES3.

Figure 5.6 shows that ES1 has the lowest runtime. ES3 and ES4 finish close to this eight-worker static reference, and the runtime of ES4 is only 0.197% higher than ES3. ES2 takes longer because it keeps four workers schedulable throughout the execution. ES5 has the longest runtime and increases it by 29.628% relative to ES3.

Figure 5.7 shows the opposite ordering for the two static references. ES1 consumes the most processor-domain energy, while ES2 consumes the least. Relative to ES3, ES4 reduces processor-domain energy by 4.566%. ES5 reduces it by 5.022%, which is only 0.456 percentage points more than ES4 despite its much larger runtime penalty.

The static settings provide reference points for interpreting the dynamic policies. ES1 achieves the shortest runtime but also has the highest processor-domain energy among ES1–ES5. ES2 reduces energy at the cost of a longer runtime. Nevertheless, under the evaluated workload, maintaining four schedulable workers throughout the execution results in both a shorter runtime and lower processor-domain energy than the power-oriented policy. This comparison does not establish that four static workers are preferable for other workloads, but it shows that the current power-oriented policy does not identify the most favorable runtime–energy operating point in this experiment.

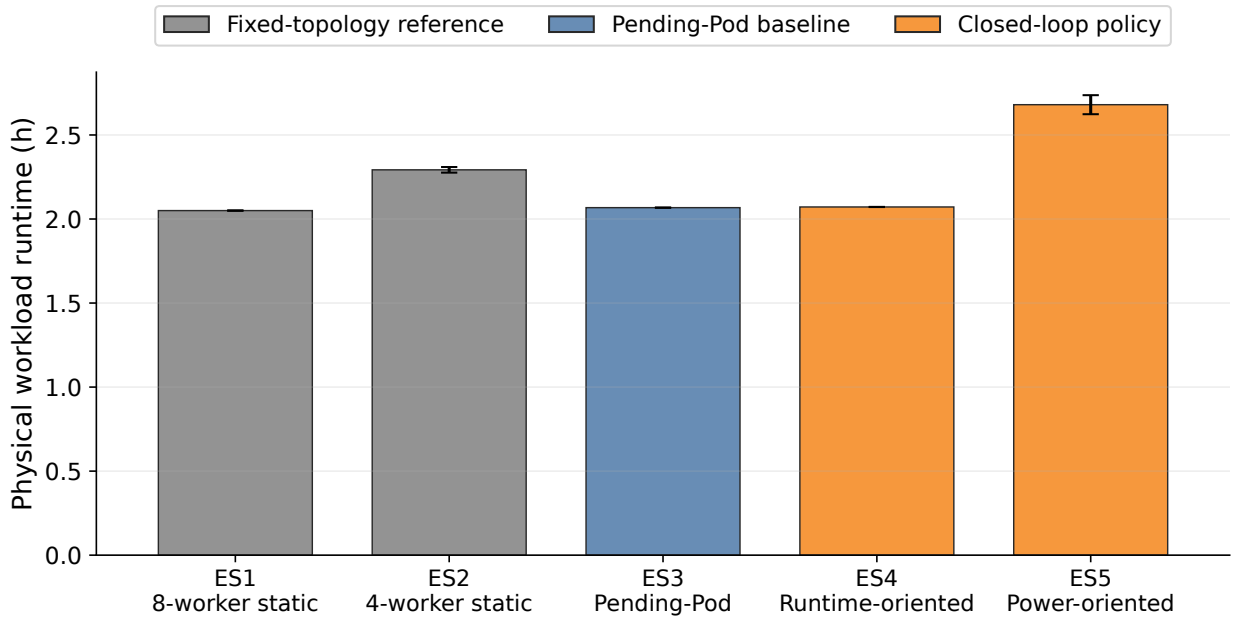

Figure 5.6: Physical workload runtime under ES1–ES5.

Setting	Strategy	Runtime (h)	Runtime change (%)	Energy (MJ)	Energy change (%)
ES1	Eight-worker static	2.0501 ± 0.0013	–	0.145933 ± 0.000082	–
ES2	Four-worker static	2.2926 ± 0.0167	–	0.127691 ± 0.000143	–
ES3	Pending-Pod baseline	2.0676 ± 0.0006	0.000	0.138965 ± 0.000157	0.000
ES4	Runtime-oriented policy	2.0717 ± 0.0005	+0.197	0.132620 ± 0.001305	−4.566
ES5	Power-oriented policy	2.6802 ± 0.0566	+29.628	0.131986 ± 0.000129	−5.022

Table 5.8: Physical runtime and processor-domain energy under ES1–ES5. Percentage changes are calculated from the setting means relative to ES3.

5.4.2 Runtime-Energy Trade-Off

The runtime–energy trade-off is assessed by interpreting the two physical outcomes jointly rather than combining them into an arbitrary weighted score. Relative to ES3, ES4 reduces processor-domain energy by 4.566% for a 0.197% runtime increase. ES5 saves only an additional 0.456 percentage points of energy, but increases runtime by 29.628%. The runtime-oriented policy therefore offers the stronger balance under the evaluated workload.

This outcome can be interpreted through the distinction between the implemented policy objective and the evaluated physical outcome. The power-oriented policy ranks candidates by mean simulated processor power instead of by physical processor-domain energy. Inspection of the decision traces indicates that it tends to select candidates with fewer schedulable workers. Lower schedulable capacity can reduce processor power, but it can also reduce parallelism and increase Job queueing. Since energy accumulates over the full

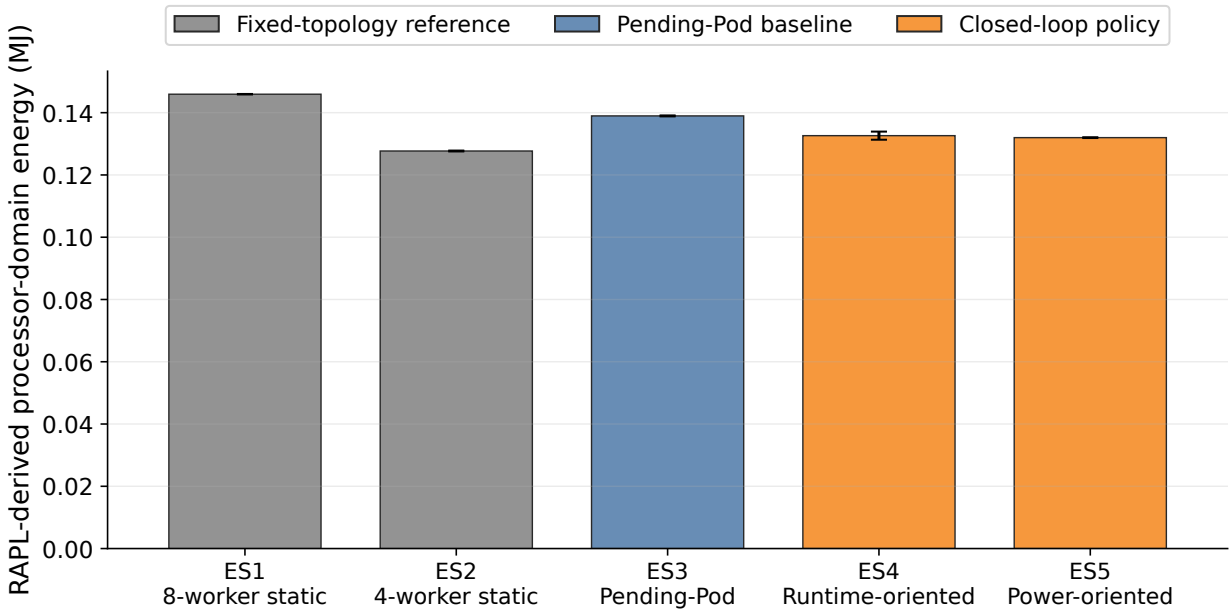

Figure 5.7: RAPL-derived processor-domain energy under ES1–ES5.

workload interval, the longer execution offsets much of the reduction in power. Minimizing mean simulated power is therefore not necessarily equivalent to minimizing physical processor-domain energy.

The power-oriented result exposes a limitation of the evaluated policy configuration rather than a general limitation of simulator-informed control. The implemented ranked policy uses tolerance-aware lexicographic ordering. In the evaluated power-oriented configuration, mean simulated processor power is the primary objective, while runtime is used only to break ties under the configured power tie tolerance. The policy does not impose an explicit limit on acceptable runtime degradation. A larger power tolerance could allow runtime to influence more candidate comparisons through the existing ranked-policy mechanism, whereas an explicit runtime-degradation constraint would require an extension. Neither alternative was evaluated in this thesis. In addition, the controller changes only node schedulability, while all eight workers remain provisioned and monitored. The measured reductions therefore reflect differences in RAPL-derived processor-domain energy during workload execution, not energy savings from deprovisioning or powering down workers.

RQ3 is answered in bounded terms. Under the evaluated workload and platform, the closed loop demonstrates decision-support value, with the runtime-oriented policy providing the stronger practical runtime–energy trade-off. The power-oriented result further

shows that this value depends on how the policy prioritizes its objectives and configures their tolerances.

5.5 Threats to Validity

Construct validity. Construct validity is limited by how simulation fidelity and energy outcomes are measured. Fidelity is assessed through Job progress, timing, processor-power magnitude, and processor-power trend, so the findings concern fit-for-purpose fidelity for these behaviors rather than the accuracy of every simulator output. The lag-tolerant correlation metric may overstate local agreement because each workload segment can select a different best lag instead of using one consistent temporal alignment across the complete execution. Therefore, it is interpreted alongside globally aligned and magnitude-based metrics. Energy measurements cover only RAPL-derived processor domains. Moreover, marking workers as unschedulable affects only future Pod placement and does not evict or migrate running Pods. The reported outcomes therefore characterize the evaluated scheduling-capacity control and must not be interpreted as total machine or cluster energy savings.

Internal validity. Internal validity may be affected by residual variation and monitoring noise arising from virtualization, Kubernetes scheduling, background activity, telemetry sampling, and metric collection. Using the same fixed-seed workload and repeating each experimental setting three times reduces these effects but does not eliminate them. In addition, the closed-loop settings include a backlog-relief rule that can make another worker schedulable independently of simulator-informed candidate ranking. The resulting physical outcomes therefore represent the complete implemented control strategy and cannot be attributed solely to the simulator-based policy.

External validity. External validity is limited by the use of one virtualized Kubernetes testbed, one OpenDC-based simulator, a homogeneous eight-worker pool, and one fixed realization of a synthetic CPU-bound finite-Job workload. The evaluated action changes worker schedulability within the existing pool rather than provisioning or migrating machines. Therefore, the findings demonstrate feasibility and limited decision-support value for the tested environment, not universal policy performance. They may not generalize to heterogeneous nodes, continuously running services, network- or memory-intensive workloads, larger continuum deployments, other simulators, or broader control actions.

6

Related Work

This chapter positions the proposed trace-based closed-loop digital twin against related work. The thesis does not contribute a new Kubernetes simulator or optimizer. Its contribution is a general architecture for the digital continuum that makes the contracts between loop responsibilities explicit. Traces have defined semantics, simulation evidence is linked to the state that produced it, calibration is scoped to that evidence, and only validated results can reach bounded actuation. The digital entity remains replaceable, so analytical, learned, simulated, or hybrid backends can be used when they expose compatible evidence and decision interfaces. The discussion is organized around digital-twin architectures, digital-entity realizations, state and result lineage, and calibration and fidelity assessment.

6.1 Digital Twin Architectures

OpenDT is the closest architectural and technical foundation for this thesis. It connects continuous datacenter telemetry with trace-driven OpenDC simulation, self-calibration, and SLO-oriented feedback under human supervision (54). It is directly relevant because the prototype developed in this thesis reuses OpenDC and adapts OpenDT's trace-replay and calibration mechanisms to observations from live Kubernetes execution. However, the evaluated OpenDT prototype demonstrates telemetry ingestion, simulation, and calibration, while automated steering of the physical infrastructure remains future work. This thesis extends that foundation by completing the operational path from observed state to state-derived candidate generation, explicit policy-based selection, and bounded actuation on the physical system.

KubeTwin provides a complementary digital-twin framework centered on Kubernetes deployments (69). Its discrete-event model represents services, replicas, scheduling, autoscaling, application workflows, request processing, and network behavior across edge and cloud resources. It supports risk-free what-if analysis of Kubernetes configurations and management policies, including replica-count selection and autoscaling reenactment. The main difference from this thesis is the architectural scope. KubeTwin develops a detailed digital twin of Kubernetes behavior, whereas this thesis designs a general closed-loop digital-twin architecture for the digital continuum. The proposed architecture treats endpoint, edge, and cloud resources as one physical execution environment, hides platform-specific mechanisms behind observation and actuation boundaries, and keeps the digital entity replaceable. Kubernetes and OpenDC instantiate the prototype, but they do not define the scope of the architecture. KubeTwin consequently contributes deeper Kubernetes-specific representation, while this thesis contributes the general responsibilities and interfaces needed to operate a digital twin across the digital continuum.

6.2 Digital Entity Realizations

Related systems show that the digital entity in a digital twin is an architectural role rather than a prescribed implementation. KAPETÁNIOS learns changed application behavior from production and canary measurements, simulates alternative Horizontal Pod Autoscaler (HPA) configurations, and applies the selected configuration to a production Kubernetes cluster (70). ATOM uses a Layered Queueing Network model to select microservice replica counts and CPU shares, while COSCO combines learned quality-of-service (QoS) models with co-simulation for task placement and container migration (71, 72). These systems demonstrate that analytical, learned, simulated, and hybrid software representations can all provide the digital reasoning needed for runtime resource-management decisions.

This thesis therefore does not claim that trace-based simulation is the only, or universally preferable, realization of a digital entity. Instead, it addresses a gap that arises when trace-based simulation is chosen. The interface between observed execution and the simulator must define what the trace means, not only how it is encoded. The trace contract developed in this thesis specifies the semantics needed to replay and compare execution, including the interpretation of timestamps and execution events, the association of resource measurements with workload activity, and the topology and configuration context under which an observation was produced. Prior systems primarily contribute a model, optimizer, or

specialized adaptation loop. They do not make simulator-compatible trace semantics the central interface of a general digital-twin architecture. By defining this interface explicitly, this thesis enables trace-derived workloads to be replayed, calibrated, and evaluated under alternative digital-continuum configurations without treating the modeling backend as fixed.

6.3 State and Result Lineage

SimuScale is the closest resource-control system to the end-to-end path implemented in this thesis (73). It captures live state from a Kubernetes-based serverless edge platform, reconstructs that state in a trace-driven simulator, evaluates alternative autoscaler parameters, and applies the selected parameter to the running orchestration component. SimuScale reduces the risk of outdated input by bounding the duration of its optimization process. This thesis addresses the same risk through explicit state and result identifiers, state lineage, rejection of reports that no longer match the currently observed state, and a freshness limit on returned reports.

6.4 Calibration and Fidelity Assessment

At the calibration level, OpenDT is again the closest work. Its self-calibrator evaluates candidate power-model parameters through short simulations, compares the resulting power estimates with recent physical telemetry, and selects the parameter that minimizes the error (54). This mechanism provides the basis for the windowed calibration implemented in this thesis. Yet the reported OpenDT experiment considers one recorded datacenter topology and does not evaluate how calibration evidence should be scoped when alternative resource topologies are simulated.

Muñoz et al. provide a more specific method for assessing behavioral fidelity between physical and digital twins using trace alignment (74). Their method represents behavior as sequences of state snapshots and aligns equivalent states without requiring them to occur at identical timestamps. The resulting matches, mismatches, gaps, and trace distances provide evidence about temporal displacement and missing or divergent behavior. This is relevant when corresponding physical and simulated behavior occurs at different times. However, the method performs offline validation rather than simulator calibration or runtime control, and its comparison thresholds and alignment parameters remain dependent on the system and intended use.

6. RELATED WORK

Relative to these approaches, this thesis combines topology-scoped calibration with a multi-dimensional assessment of fit-for-purpose fidelity. A bounded search calibrates one OpenDC CPU power-model factor from recent physical evidence, and the selected value is stored with the topology that produced that evidence. Separate metrics assess workload progress, Job timing, processor-power magnitude, and processor-power trend because improving one power-model parameter does not establish fidelity in every decision-relevant dimension. Trace alignment could provide additional diagnostic evidence about temporal displacement and localized disagreement, but it remains outside the scope of this thesis.

7

Conclusion

This thesis focused on the architectural design of a closed-loop digital twin for resource management in the digital continuum. It proposed a reference architecture that connects physical observation, trace-based simulation, calibration, decision making, and bounded actuation into an evidence-driven feedback loop. To demonstrate how this architecture can be realized, a cloud-resource-level prototype was implemented using Kubernetes, OpenDC, and mechanisms adapted from OpenDT. The evaluation demonstrates the feasibility of the proposed closed-loop design and shows that its decision-support value depends on simulator fidelity, calibration scope, and policy design.

7.1 Answering the Research Questions

RQ1: Closed-loop architecture. RQ1 asked how trace-based simulation can mirror physical execution and support feedback-driven adaptation. The proposed architecture separates the system into a physical space, a closed-loop control plane, and a digital space. Observation and actuation boundaries isolate the physical system, while the simulator and calibrator provide evidence without directly controlling it. The control plane coordinates candidate generation, simulation, validation, selection, and actuation. Candidates and outcomes remain linked to their originating physical state, only supported actions can be applied, and a no-operation path remains available. These properties make the feedback path state-valid, inspectable, and bounded. Kubernetes and OpenDC instantiate the prototype, while explicit interfaces keep the principal components conceptually replaceable.

RQ2: Operationalizing the trace-to-simulation-to-action workflow. RQ2 is addressed through an operational trace contract, topology-aware calibration, and a state-

bound path from candidate evaluation to physical action.

RQ2a: Operational trace contract. A closed-loop trace must preserve workload, telemetry, system snapshot, and lineage. Workload records retain task arrival, duration, declared capacity, and sampled CPU demand for replay. Telemetry records retain timestamped physical measurements used as calibration evidence. System snapshots describe the observed resource configuration and the capacity bound used to construct what-if candidates. State, batch, candidate, and report identifiers determine whether returned evidence still applies to the current physical state. The contract therefore specifies timestamps, units, event granularity, and configuration context instead of only a file format, while platform-specific translation remains localized at the physical and simulator boundaries.

RQ2b: Topology-aware calibration. The prototype calibrates OpenDC's CPU power model against recent physical processor-power measurements. A bounded grid search replays the workload under the observed topology and selects the factor with the lowest aligned error over a shared interval. The calibration factor and its evidence context are stored by topology and can be reused when that topology reappears. This prevents evidence from one worker configuration from being treated as direct evidence for another. However, the evaluation shows that this method produces mixed rather than uniform fidelity improvement. It adjusts one power-model parameter and does not calibrate execution timing, scheduling, queueing, processor performance, or resource sharing.

RQ2c: State-bound decision and actuation. Each decision cycle begins with a single observed system snapshot. The candidate generator creates alternative candidates and a no-operation candidate, all evaluated using the same workload and interval. Returned reports retain state, batch, and candidate lineage. Stale, incomplete, unknown, or state-mismatched reports are rejected before policy-based ranking, and the state is checked again before actuation. The controller then translates the selected abstract decision into a bounded operation supported by the physical execution environment, after which the physical system is re-observed. Simulator output is therefore evidence rather than a command.

RQ3: Decision-support value. Under the evaluated workload and platform, the closed loop demonstrates decision-support value within this scope. The runtime-oriented policy

keeps runtime close to the pending-Pod baseline while reducing RAPL-derived processor-domain energy, giving it the stronger practical runtime–energy trade-off. The power-oriented policy reduces energy only modestly further but introduces a large runtime penalty because it tends to select fewer schedulable workers. Lower capacity reduces power but also reduces parallelism and increases queueing, so the longer execution offsets much of the reduction. Minimizing mean simulated power is therefore not equivalent to minimizing physical processor-domain energy. The value of simulator-informed control depends on the configuration of the decision policy. The findings concern processor-domain energy during workload execution instead of total-machine or total-cluster energy.

RQ1 defines the responsibilities and interfaces needed to complete the loop, RQ2 demonstrates their end-to-end realization, and RQ3 evaluates the resulting physical decisions. The central contribution is an evidence-driven methodology in which simulation results remain linked to their physical context and reach the physical system only through validation and bounded control.

7.2 Limitations and Future Work

7.2.1 Limitations

OpenDC provides comparative evidence instead of an exact reproduction of Kubernetes execution. It does not model every scheduler constraint, pending-Pod behavior, control-plane delay, or resource-sharing effect. Under constrained capacity, it underestimates tail queueing and total slowdown, while processor-power magnitude remains imperfect. Calibration is limited to one CPU power-model factor, has mixed effects across the evaluated fidelity measures, and its error is not incorporated into candidate ranking.

The architecture targets the digital continuum, but the prototype evaluates only a homogeneous cloud-resource layer. It changes worker schedulability within an already provisioned pool and does not provision or power down machines. Marking a worker unschedulable affects only future Pod placement. A backlog-relief rule can also change schedulability independently of simulator ranking, so the outcomes characterize the complete implemented strategy rather than full infrastructure autoscaling.

The evaluation uses one virtualized testbed, one simulator, and one realization of a synthetic CPU-bound finite-Job workload. Measurements may be affected by virtualization, scheduling, background activity, telemetry sampling, and monitoring noise. RAPL covers processor domains rather than complete machines, and no statistical significance tests are

performed. The findings therefore demonstrate feasibility and decision-support value for the tested environment instead of general policy performance.

7.2.2 Future Work

Calibration could be extended to multiple parameters and metrics, including power, task progress, runtime, queueing, and utilization. Trace-alignment methods could distinguish timing displacement from magnitude disagreement. Fidelity thresholds or uncertainty estimates could also be incorporated into candidate selection so that weakly supported improvements are rejected. These mechanisms are proposed extensions and are not implemented in the prototype.

Future policies could estimate energy over the complete predicted workload interval, or impose an explicit limit on runtime degradation. The decision space could be extended to provisioning, placement, migration, Pod resource limits, admission control, or processor-frequency management, with additional platform-specific safety checks.

Broader validation should include heterogeneous cloud, edge, and endpoint resources, network and data-movement effects, continuously running services, and workloads that stress memory, storage, and communication. Larger deployments, additional workload realizations, more repetitions, and alternative digital-entity backends would provide stronger evidence about policy robustness and architectural portability.

Overall, this thesis shows that a trace-based closed-loop digital twin can connect physical observation to simulator-informed action without allowing the simulator to control the physical system directly. Explicit trace semantics, topology-aware calibration, lineage, validity checks, policy-based selection, and bounded actuation provide a feasible foundation for inspectable decision support within the evaluated scope.

References

[1] Md Firoj Ali and Rafiqul Zaman Khan. **Distributed computing: An overview**. *International journal of advanced networking and applications*, **7**(1):2630, 2015. 3

[2] Song Fu. **Failure-aware resource management for high-availability computing clusters with distributed virtual machines**. *Journal of Parallel and Distributed Computing*, **70**(4):384–393, 2010. 3

[3] Noel Crespi, Adam T. Drobot, and Roberto Minerva, editors. *The Digital Twin*. Springer International Publishing, Jan 2023. 3

[4] Yan Zhang. *Digital twin: architectures, networks, and applications*. Springer, 2024. 3

[5] Larry Schmitt and David Copps. *The Business of Digital Twins*, pages 21–63. Springer International Publishing, Cham, 2023. 3

[6] Visual Components. **Understanding digital twins in manufacturing - Visual Components**, Feb 2025. 3

[7] CADD Centre. **What are Digital Twins in Civil Engineering and How to Create them?**, Jun 2023. 3

[8] Markets and Markets. **Digital Twin Market Size Global forecast to 2026**, Jul 2023. 3

[9] Matilda Adolphsen. **Digital Twin ROI: Breaking Down Cost Reductions with Real Numbers | Simio**, Feb 2026. 3

[10] Max Blanchet. *The Dimension of Markets for the Digital Twin*, pages 65–96. Springer, 06 2023. 4

REFERENCES

[11] Auday Al-Dulaimy, Matthijs Jansen, Bjarne Johansson, Animesh Trivedi, Alexandru Iosup, Mohammad Ashjaei, Antonino Galletta, Dragi Kimovski, Radu Prodan, Konstantinos Tserpes, George Kousiouris, Chris Giannakos, Ivona Brandic, Nawfal Ali, André B. Bondi, and Alessandro V. Papadopoulos. **The computing continuum: From IoT to the cloud**. *Internet of Things*, **27**:101272, 2024. 4

[12] Daniel Rosendo, Alexandru Costan, Patrick Valduriez, and Gabriel Antoniu. **Distributed intelligence on the Edge-to-Cloud Continuum: A systematic literature review**. *Journal of Parallel and Distributed Computing*, **166**:71–94, 2022. 4, 5

[13] Caihong Kai, Hao Zhou, Yibo Yi, and Wei Huang. **Collaborative Cloud-Edge-End Task Offloading in Mobile-Edge Computing Networks With Limited Communication Capability**. *IEEE Transactions on Cognitive Communications and Networking*, **7**(2):624–634, 2021. 4

[14] Sungju Lee, Heegon Kim, and Yongwha Chung. **Power-Time Tradeoff of Parallel Execution on Multi-core Platforms**. In James J. (Jong Hyuk) Park, Hojjat Adeli, Namje Park, and Isaac Woungang, editors, *Mobile, Ubiquitous, and Intelligent Computing*, pages 157–163, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg. 4

[15] M. Etinski, J. Corbalan, J. Labarta, and M. Valero. **Understanding the future of energy-performance trade-off via DVFS in HPC environments**. *Journal of Parallel and Distributed Computing*, **72**(4):579–590, 2012. 4

[16] Stefan Mihai, Mahnoor Yaqoob, Dang V Hung, William Davis, Praveer Towakel, Mohsin Raza, Mehmet Karamanoglu, Balbir Barn, Dattaprasad Shetve, Raja V Prasad, et al. **Digital twins: A survey on enabling technologies, challenges, trends and future prospects**. *IEEE Communications Surveys & Tutorials*, **24**(4):2255–2291, 2022. 5

[17] Chenhao Wu, Zhexin Cui, Qian Xia, Jiguang Yue, and Feng Lyu. **An overview of digital twin technology for power electronics: State-of-the-art and future trends**. *IEEE Transactions on Power Electronics*, 2025. 5

[18] Hanpeng Hu, Dan Wang, and Chuan Wu. **Distributed Machine Learning through Heterogeneous Edge Systems**. *CoRR*, **abs/1911.06949**, 2019. 5

[19] Aidan Fuller, Zhong Fan, Charles Day, and Chris Barlow. **Digital Twin: Enabling Technologies, Challenges and Open Research**. *IEEE Access*, **8**:108952–108971, 2020. 5, 13

[20] Jiaqi Wen, Bogdan Gabrys, and Katarzyna Musial. **Toward digital twin oriented modeling of complex networked systems and their dynamics: A comprehensive survey**. *IEEE Access*, **10**:66886–66923, 2022. 6, 14

[21] Igiri Onaji, Divya Tiwari, Payam Soulatiantork, Boyang Song, and Ashutosh Tiwari. **Digital twin in manufacturing: conceptual framework and case studies**. *International journal of computer integrated manufacturing*, **35**(8):831–858, 2022. 6

[22] 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**. *CoRR*, **abs/1902.05416**, 2019. 8

[23] Richard Wesley Hamming. **The art of doing science and engineering: learning to learn**. 1998. 8

[24] Ken Peffers, Tuure Tuunanen, Marcus Rothenberger, and S. Chatterjee. **A design science research methodology for information systems research**. *Journal of Management Information Systems*, **24**:45–77, 01 2007. 8

[25] Raj Jain. *The Art of Computer Systems Performance Analysis: Techniques For Experimental Design, Measurement, Simulation, and Modeling, NY: Wiley*. 4 1991. 9

[26] Gernot Heiser. **Systems Benchmarking Crimes**, 2019. 9

[27] John Ousterhout. **Always measure one level deeper**. *Commun. ACM*, **61**(7):74–83, June 2018. 9

[28] Sonja Bezjak, April Clyburne-Sherin, Philipp Conzett, Pedro Fernandes, Edit Görögh, Kerstin Helbig, Bianca Kramer, Ignasi Labastida, Kyle Niemeyer, Fotis Psomopoulos, Tony Ross-Hellauer, René Schneider, Jon Tennant, Ellen Verbakel, Helene Brinken, and Lambert Heller. *Open Science Training Handbook*. Zenodo, April 2018. 9

[29] Mark D. Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, et al. **The FAIR Guiding Principles for Scientific Data Management and Stewardship**. *Scientific Data*, **3**:160018, 2016. 9

[30] Emery D. Berger, Stephen M. Blackburn, Matthias Hauswirth, and Michael W. Hicks. **A Checklist Manifesto for Empirical Evaluation: A Preemptive Strike Against a Replication Crisis in Computer Science**. ACM SIGPLAN Blog, August 2019. Published 28 August 2019. 9

[31] Alexandru Uta, Alexandru Custura, Dmitry Duplyakin, Ivo Jimenez, Jan Rellermeyer, Carlos Maltzahn, Robert Ricci, and Alexandru Iosup. **Is big data performance reproducible in modern cloud networks?** In *Proceedings of the 17th Usenix Conference on Networked Systems Design and Implementation*, NSDI'20, page 513–528, USA, 2020. USENIX Association. 9

[32] Fei Tao, He Zhang, Ang Liu, and A. Y. C. Nee. **Digital Twin in Industry: State-of-the-Art**. *IEEE Transactions on Industrial Informatics*, **15**(4):2405–2415, April 2019. 13

[33] 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. 13

[34] Constantin Cronrath, Ludvig Ekström, and Bengt Lennartson. **Formal Properties of the Digital Twin – Implications for Learning, optimization, and Control**. In *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*, pages 679–684, Aug 2020. 14

[35] Shohin Aheleroff, Xun Xu, Ray Y. Zhong, and Yuqian Lu. **Digital Twin as a Service (DTaaS) in Industry 4.0: An Architecture Reference Model**. *Advanced Engineering Informatics*, **47**:101225, 2021. 15

[36] Tsega Y. Melesse, Valentina Di Pasquale, and Stefano Riemma. **Digital Twin Models in Industrial Operations: A Systematic Literature Review**. *Procedia Manufacturing*, **42**:267–272, 2020. International Conference on Industry 4.0 and Smart Manufacturing (ISM 2019). 15

[37] Francesco Flammini. **Digital twins as run-time predictive models for the resilience of cyber-physical systems: a conceptual framework**. *Philosophical*

Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, **379**(2207):20200369, 08 2021. 15

[38] Sergio Laso, Lorenzo Tore-Gálvez, Javier Berrocal, Carlos Canal, and Juan Manuel Murillo. **Deploying Digital Twins Over the Cloud-to-Thing Continuum**. In *2023 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–6, July 2023. 15

[39] Christian Kober, Vincent Adomat, Maryam Ahanpanjeh, Marc Fette, and Jens Wulfsberg. **Digital Twin Fidelity Requirements Model For Manufacturing**. 05 2022. 17

[40] Christian Kober, Marc Fette, and Jens P. Wulfsberg. **A Method for Calculating Optimum Digital Twin Fidelity**. *Procedia CIRP*, **120**:1155–1160, 2023. 56th CIRP International Conference on Manufacturing Systems 2023. 17

[41] David Jones, Chris Snider, Aydin Nassehi, Jason Yon, and Ben Hicks. **Characterising the Digital Twin: A systematic literature review**. *CIRP Journal of Manufacturing Science and Technology*, **29**:36–52, 2020. 17

[42] Paula Muñoz. **Measuring the fidelity of digital twin systems**. In *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, MODELS '22, page 182–188, New York, NY, USA, 2022. Association for Computing Machinery. 17

[43] David J. Lilja. *Measuring computer performance : a practitioner's guide*. Cambridge University Press, Cambridge, 1st ed. edition, 2000. 17

[44] Lieven Eeckhout. *Analytical Performance Modeling*, pages 31–47. Springer International Publishing, Cham, 2010. 17

[45] Vitor Lamela, Helder Fontes, Tiago Oliveira, Jose Ruela, Manuel Ricardo, and Rui Campos. **Repeatable and Reproducible Wireless Networking Experimentation through Trace-based Simulation**. In *2019 International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pages 53–58, 2019. 17

[46] Guodong Shao, Sanjay Jain, Christoph Laroque, Loo Hay Lee, Peter Lendermann, and Oliver Rose. **Digital Twin for Smart Manufacturing:**

The Simulation Aspect. In *2019 Winter Simulation Conference (WSC)*, pages 2085–2098, Dec 2019. 18

[47] Zwerus Cornelis Roza. *Simulation fidelity theory and practice: A unified approach to defining, specifying and measuring the realism of simulations*. Delft University Press, 2004. 18

[48] Katherine Campbell. **Statistical calibration of computer simulations**. *Reliability Engineering & System Safety*, **91**(10):1358–1363, 2006. The Fourth International Conference on Sensitivity Analysis of Model Output (SAMO 2004). 18, 19

[49] Laurens Versluis, Roland Mathá, Sacheendra Talluri, Tim Hegeman, Radu Prodan, Ewa Deelman, and Alexandru Iosup. **The Workflow Trace Archive: Open-Access Data From Public and Private Computing Infrastructures**. *IEEE Trans. Parallel Distributed Syst.*, **31**(9):2170–2184, 2020. 20

[50] J.O. Kephart and D.M. Chess. **The vision of autonomic computing**. *Computer*, **36**(1):41–50, 2003. 21

[51] Rogério de Lemos, Holger Giese, Hausi A Müller, Mary Shaw, J Andersson, M Litoiu, B Schmerl, G Tamura, NM Villegas, T Vogel, et al. **Software Engineering for Self-Adaptive Systems II**. *Lecture Notes in Computer Science*, **7475**, 2010. 21

[52] Autonomic Computing et al. **An architectural blueprint for autonomic computing**. *IBM White paper*, **31**(2006):1–6, 2006. 21

[53] 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 *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pages 455–464, 2021. 36, 38

[54] 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 *Companion of the 17th*

ACM/SPEC International Conference on Performance Engineering, ICPE Companion '26, page 142–147, New York, NY, USA, 2026. Association for Computing Machinery. 36, 46, 75, 77

[55] Matthijs Jansen, Linus Wagner, Animesh Trivedi, and Alexandru Iosup. **Continuum: Automate Infrastructure Deployment and Benchmarking in the Compute Continuum**. In *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering*, ICPE '23 Companion, page 181–188, New York, NY, USA, 2023. Association for Computing Machinery. 37

[56] The Kubernetes Authors. **Kubernetes Documentation: Overview**. Accessed 24 July 2026. 37, 38

[57] Prometheus Authors. **Prometheus Documentation: Overview**. Accessed 24 July 2026. 38

[58] Hubblo. **Scaphandre Documentation**. Accessed 24 July 2026. 38

[59] Apache Software Foundation. **Apache Kafka Documentation**. Accessed 24 July 2026. 38

[60] PostgreSQL Global Development Group. **PostgreSQL Documentation**. Accessed 24 July 2026. 38

[61] Apache Software Foundation. **Apache Parquet Documentation: Overview**. Accessed 24 July 2026. 38

[62] Docker, Inc. **Docker Compose Documentation**. Accessed 24 July 2026. 38

[63] Sebastián Ramírez. **FastAPI Documentation**. Accessed 24 July 2026. 38

[64] Grafana Labs. **Grafana Documentation: Dashboards**. Accessed 24 July 2026. 38

[65] Thimo Wuttge. **BenchFrame: A Framework for Benchmarking Power Monitoring Tools**, August 2025. Accessed 24 July 2026. 39

[66] Guillaume Raffin and Denis Trystram. **Dissecting the Software-Based Measurement of CPU Energy Consumption: A Comparative Analysis**. *IEEE Transactions on Parallel and Distributed Systems*, **36**(1):96–107, 2025. 43

[67] Kubernetes Authors. **Node Autoscaling**, 2026. Accessed: 2026-08-02. 58

[68] Guanghe Xie. **From Reality to Simulation: A Trace-Driven Closed-Loop Evaluation Methodology for Cloud Datacenters**, November 2025. Accessed 3 Aug 2026. 61

[69] Davide Borsatti, Walter Cerroni, Luca Foschini, Genady Ya Grabarnik, Lorenzo Manca, Filippo Poltronieri, Domenico Scotece, Larisa Shwartz, Cesare Stefanelli, Mauro Tortonesi, and Mattia Zaccarini. **KubeTwin: A Digital Twin Framework for Kubernetes Deployments at Scale**. *IEEE Transactions on Network and Service Management*, **21**(4):3889–3903, 2024. 76

[70] Johannes Zerwas, Patrick Krämer, Răzvan-Mihai Ursu, Navidreza Asadi, Phil Rodgers, Leon Wong, and Wolfgang Kellerer. **KAPETÁNIOS: Automated Kubernetes Adaptation through a Digital Twin**. In *2022 13th International Conference on Network of the Future (NoF)*, pages 1–3, 2022. 76

[71] Alim Ul Gias, Giuliano Casale, and Murray Woodside. **ATOM: Model-Driven Autoscaling for Microservices**. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 1994–2004, 2019. 76

[72] Shreshth Tuli, Shivananda R. Poojara, Satish N. Srirama, Giuliano Casale, and Nicholas R. Jennings. **COSCO: Container Orchestration Using Co-Simulation and Gradient Based Optimization for Fog Computing Environments**. *IEEE Transactions on Parallel and Distributed Systems*, **33**(1):101–116, 2022. 76

[73] Philipp Raith, Stefan Nastic, and Schahram Dustdar. **SimuScale: Optimizing Parameters for Autoscaling of Serverless Edge Functions Through Co-Simulation**. In *2024 IEEE 17th International Conference on Cloud Computing (CLOUD)*, pages 305–315, 2024. 77

[74] Paula Muñoz, Manuel Wimmer, Javier Troya, and Antonio Vallecillo. **Measuring the Fidelity of a Physical and a Digital Twin Using Trace Alignments**. *IEEE Transactions on Software Engineering*, **50**(12):3122–3145, Dec 2024. 77

Appendix A

Reproducibility

A.1 Abstract

This appendix describes the artifacts used to reproduce the implementation and experiments presented in this thesis. The closed-loop digital twin prototype, collected experiment data, analysis code, and Kubernetes workload generator are publicly available.

A.2 Artifact check-list (meta-information)

- **Program:** Python, Docker Compose, OpenDC, Kubernetes
- **Data set:** Generated Kubernetes workload and collected experiment data
- **Run-time environment:** Linux/Ubuntu, Continuum, QEMU/KVM, Kubernetes
- **Hardware:** Five x86-64 physical hosts with Intel Xeon Silver 4210R processors; one control-plane VM and eight worker VMs, each with 8 virtual CPU cores and 32 GB memory
- **Metrics:** Makespan, task progress and timing, processor-power fidelity, physical runtime, and RAPL-derived processor-domain energy
- **Output:** Parquet data, analysis tables, and figures
- **Experiments:** ES1–ES10 described in Chapter 5
- **How much disk space required (approximately)?:** 10 GB
- **How much time is needed to prepare workflow (approximately)?:** 1–2 hours
- **How much time is needed to complete experiments (approximately)?:** About 75 hours
- **Publicly available?:** Yes

A.3 Description

A.3.1 How to access

- The closed-loop digital twin source code: `https://github.com/EDChui/closed-loop-opendt`
- The collected experiment data: `https://github.com/EDChui/closed-loop-opendt/releases/tag/v1.0.0`
- The workload generator source code: `https://github.com/EDChui/k8s-workload-generator`

A.3.2 Hardware dependencies

The deployment requires Linux hosts with QEMU/KVM virtualization support. Reproducing the processor-domain energy measurements additionally requires Intel RAPL access through Scaphandre.

A.3.3 Software dependencies

The experiment environment uses Continuum to provision the virtualized testbed and Kubernetes (`v1.27.16`) as the resource manager. Prometheus and Scaphandre provide observability and processor-domain energy measurements. The prototype deployment requires Docker Compose, Make, Python 3.11 or newer, and `uv`. The OpenDC binary (`v2.4f`) is included in the source code. The workload generator uses Python 3.8.10. Exact service configuration and testbed-specific setup are documented in the repository README.

A.4 Installation

Clone the repositories, check out the `v1.0.0` tag of `closed-loop-opendt`, and follow its README and `reproducibility-capsule/setup.md`. The prototype setup uses `make setup` followed by `make up`. The workload generator is installed from its `requirements.txt` and executed using `python3 src/cli.py`.

A.5 Experiment workflow

For analysis-only reproduction, download the collected experiment data and run the analysis scripts in the reproducibility capsule to regenerate the tables and figures in Chapter 5. For end-to-end reproduction, provision the Continuum/Kubernetes testbed, start

the closed-loop services, execute the fixed-seed 370-Job staged workload under ES1–ES10 with three repetitions per setting, and run the same analysis.