

Vrije Universiteit Amsterdam



University of Amsterdam



Master Thesis

---

# Scalability in System-Level Simulation Environments for Distributed Cyber-Physical Systems

---

**Author:** Herman Kelder

*1st supervisors:* Prof. dr. A.D. Pimentel  
Prof. dr. B. Akesson  
*daily supervisor:* M. Herget, M.Sc.  
*2nd reader:* Prof. dr. R.V. van Nieuwpoort

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

July 5, 2023

---

*“We are just an advanced breed of monkeys on a minor planet of a very average star.  
But we can understand the Universe. That makes us something very special.”*

– Stephen Hawking, *Der Spiegel* 1988

## Abstract

Industrial Cyber-Physical Systems (CPS) drive industry sectors worldwide, combining physical and software components into sophisticated interconnected systems. Distributed CPS (dCPS) further enhance these systems by interconnecting multiple distributed subsystems through intricate, complex networks. Researchers and industrial designers need to carefully consider various design options that have the potential to impact system behaviour, cost, and performance during the development of dCPS. However, the increased size and complexity present manufacturing companies with new challenges when designing their next-generation machines. Furthermore, objectively evaluating these machines' vast number of potential arrangements can be resource-intensive. One of the approaches designers can utilise to aid themselves with early directions in the design process is Design Space Exploration (DSE). Nevertheless, the vast amount of potential design points (a single system configuration) in the design space (collection of all possible design points) poses a significant challenge to scalably and efficiently reach an exact or reasonable solution during the design process.

This thesis addresses the scalability challenge in the design process employed by researchers and designers of the next-generation complex dCPS. A baseline of understanding is constructed of the state-of-the-art, its complexity, research directions, and challenges in the context of DSE for dCPS and related research fields. To facilitate scalable and efficient DSE for dCPS, an evaluation environment is proposed, implemented, and evaluated. The research considers key design considerations for developing a distributed evaluation workflow that can dynamically be adapted to enable efficient and scalable exploration of the vast design space of complex, distributed Cyber-Physical Systems.

Evaluation of the proposed environment employs a set of system models, representing design points within a DSE process, to assess the solution and its

behaviour, performance, capability, and applicability in addressing the scalability challenge in the context of DSE for dCPS. During the evaluation, the performance and behaviour are investigated in three areas: (i) Simulation Campaign, (ii) Task Management Configuration, and (iii) Parallel Discrete-Event Simulation (PDES). Throughout the evaluation, it is demonstrated that the proposed environment is capable of providing scalable and efficient evaluation of design points in the context of DSE for dCPS. Furthermore, the proposed solution enables designers and researchers to tailor it to their environment through dynamic complex workflows and interactions, workload-level and task-level parallelism, and simulator and compute environment agnosticism.

The outcomes of this research contribute to advancing the research field towards scalable and efficient evaluation for DSE of dCPS, supporting designers and researchers developing their next-generation dCPS. Nevertheless, further research can be conducted on the impact of a system's behavioural characteristics on the performance and behaviour of the proposed solution when using the PDES methodology. Additionally, the interaction between external applications and the proposed solution could be investigated to support and enable further complex interactions and requirements.

# Contents

<b>List of Figures</b>	<b>v</b>
<b>List of Tables</b>	<b>vii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Problem Statement . . . . .	2
1.2 Objectives and Contributions . . . . .	3
1.3 Structure . . . . .	4
<b>2 Background and Related Works</b>	<b>7</b>
2.1 Cyber-Physical Systems . . . . .	7
2.2 DSE . . . . .	9
2.3 Scalability . . . . .	13
2.4 Simulation . . . . .	17
2.5 Distributed Computing . . . . .	19
2.6 Related Works . . . . .	21
2.6.1 Scaling Frameworks . . . . .	21
2.6.1.1 Simulator Capabilities . . . . .	21
2.6.1.2 Scalability Toolchains . . . . .	22
2.6.2 DSE Frameworks . . . . .	23
<b>3 Approach</b>	<b>27</b>
3.1 Simulation . . . . .	27
3.1.1 Design Points . . . . .	28
3.1.2 Scalability . . . . .	29
3.2 Distributed Computing . . . . .	30
3.2.1 Interaction . . . . .	30
3.2.2 Facilities . . . . .	31

## CONTENTS

---

3.2.3	Data Management . . . . .	33
<b>4</b>	<b>Methodology</b>	<b>35</b>
4.1	Design . . . . .	35
4.2	Implementation . . . . .	39
4.2.1	Configuration . . . . .	39
4.2.2	Design Point Object . . . . .	40
4.2.3	Distributed Computing . . . . .	41
<b>5</b>	<b>Evaluation</b>	<b>47</b>
5.1	Setup . . . . .	47
5.1.1	Platform . . . . .	47
5.1.2	Models . . . . .	48
5.2	Experiment Definition . . . . .	51
5.2.1	Experiment: Simulation Campaign . . . . .	52
5.2.2	Experiment: Worker Size . . . . .	55
5.2.3	Experiment: PDES . . . . .	56
5.3	Results . . . . .	58
5.3.1	Simulation Campaign . . . . .	59
5.3.2	Worker Size . . . . .	64
5.3.3	PDES . . . . .	65
<b>6</b>	<b>Discussion</b>	<b>69</b>
6.1	Simulation Campaign . . . . .	69
6.2	Worker Size . . . . .	72
6.3	PDES . . . . .	74
6.4	General . . . . .	76
<b>7</b>	<b>Conclusion</b>	<b>79</b>
7.1	Future Work . . . . .	82
<b>A</b>	<b>Data</b>	<b>93</b>
A.1	Experiment: Simulation Campaign . . . . .	93
A.1.1	Baselines . . . . .	93
A.1.2	Campaign . . . . .	93
A.2	Experiment: Worker Size . . . . .	95
A.3	Experiment: PDES . . . . .	97

## CONTENTS

---

A.3.1	Baselines . . . . .	97
A.3.2	Campaign: Sequential . . . . .	98
A.3.3	Campaign: PDES . . . . .	103
<b>B</b>	<b>Visualisations</b>	<b>107</b>
B.1	Experiment: Simulation Campaign . . . . .	108
B.1.1	SMT Enabled . . . . .	108
B.2	Experiment: PDES . . . . .	108
B.2.1	Sequential Campaign . . . . .	109
B.2.2	Relative Speedup . . . . .	113
<b>C</b>	<b>Models</b>	<b>119</b>
C.1	INET-LANS . . . . .	119





# List of Figures

2.1	General DSE workflow by Herget et al. [24]. . . . .	11
4.1	Scalable evaluation workflow . . . . .	36
4.2	Dask, Dask Distributed, and Dask Jobqueue overview. . . . .	43
4.3	Resource Controller internal workflow with a single Dask Worker per Compute Node. . . . .	44
4.4	Resource Controller internal workflow with four Dask Workers per Compute Node. . . . .	45
4.5	Resource Controller Dask Worker slots. . . . .	46
5.1	CQN model structure. . . . .	51
5.2	Efficiency and Speedup for strong and weak scaling with 32 evaluations as baseline for the single node configuration and the <i>Sequential</i> and <i>Sequential</i> execution time baselines. . . . .	59
5.3	Efficiency and Speedup for increasing number of simulations with both the <i>Sequential</i> and <i>Sequential Multi</i> execution time baseline. . . . .	60
5.4	Efficiency and Speedup for strong and weak scaling with 64 evaluations as baseline for the single node configuration and the <i>Sequential</i> and <i>Sequential Multi</i> execution time baselines when SMT is enabled. . . . .	61
5.5	Efficiency and Speedup for an increasing number of evaluations of the INET-LANS model with both the <i>Sequential</i> and <i>Sequential Multi</i> execution time baselines when SMT is enabled. . . . .	62
5.6	Relative speedup $S_{rel}$ for an increasing number of evaluations of the INET-LANS model with both the <i>Sequential</i> , <i>Sequential Multi</i> execution time baselines when SMT is enabled or disabled. . . . .	63

## LIST OF FIGURES

---

5.7	Evaluation time when increasing the number of workers on a single node for both weak and strong scaling with 32 evaluations as baseline for the single node configuration. . . . .	64
5.8	Speedup for strong and weak scaling with 4 PDES evaluations of a CQN model as baseline for the single node configuration and the <i>Sequential</i> , <i>Sequential Multi</i> , <i>Sequential Campaign</i> , <i>Sequential PDES</i> , and <i>Sequential PDES Multi</i> execution time baselines. . . . .	65
5.9	Efficiency for strong and weak scaling with 4 PDES evaluations of a CQN model as baseline for the single node configuration and the <i>Sequential</i> , <i>Sequential Multi</i> , <i>Sequential Campaign</i> , <i>Sequential PDES</i> , and <i>Sequential PDES Multi</i> execution time baselines. . . . .	66
5.10	Relative speedup $S_{rel}(x, y)$ comparing model $x$ against model $y$ for increasing number of PDES CQN model evaluations with the <i>Sequential Multi</i> execution time baseline. . . . .	67
5.11	Relative speedup $S_{rel}(x, y)$ comparing model $x$ against model $y$ for increasing number of PDES CQN model evaluations with the <i>Sequential PDES Multi</i> execution time baseline. . . . .	68

# List of Tables

5.1	Configuration parameters of the CQN models. . . . .	52
-----	---	----



# Introduction

In recent decades, information-technology has become increasingly intertwined with society and daily lives. At the foundation of the information-technology field are complex interconnected Cyber-Physical Systems (CPS), driving research, innovation, and operation of crucial industrial sectors such as robotics, industrial automation, avionics, space, and health industries. Deeply intertwining physical and software components, a CPS comprises a collection of subsystems to provide system-level services, where each subsystem implements a variety of functionalities (e.g., data processing, redundancy, monitoring, and control). Connecting the distributed computing infrastructure, usually incorporating heterogeneous multi-core or many-core systems, of a CPS through complex networking creates *distributed Cyber-Physical Systems* (dCPS) [2]. The complexity involved in the infrastructure, operation, and maintenance of dCPS for manufacturing companies, such as ASML, Canon Production Printing, and Philips, introduces important challenges when designing next-generation lithography scanner machines, industrial printers, and X-ray machines, respectively [43].

Researchers and industry designers regularly encounter "What-if" questions when designing and developing the next-generation of dCPS (i.e., "What if we split a subsystem into multiple smaller subsystems?", "What happens if we alter the structure of communication channels between subsystems?", and "What if we change to more efficient but less performant hardware in subsystems?"). Such design decisions have the potential to significantly affect a system's operational behaviour, performance, and costs (i.e., production and operational costs). The collection of all possible design configurations is the *Design Space*. As a result of the complexity of dCPS, a vast number of possible design configurations are created based on the design decisions present. The ramification in size and complexity of possible designs presents manufacturing companies with new challenges in

## 1. INTRODUCTION

---

designing and evaluating their next-generation machines. However, objectively evaluating the vast number of potential arrangements can be resource-intensive.

A fast expansion of possible design configurations can be mitigated by efficiently evaluating design points (one possible configuration) early on, based on Key Performance Indicators (KPIs), constraints, and other metrics. One way of analysing possible design configurations is Design Space Exploration (DSE), which aims to find an exact or reasonable solution and typically avoids evaluating the entire design space. Especially in the context of complex dCPS, exploring the entire design space is usually not possible for a time-constrained and efficient analysis. Instead of exploring the entire design space, DSE usually tries to intelligently search the design space by taking previous evaluations of design points into account. A high-accuracy approach for evaluation of a system is to build a prototype. However, depending on the system to be implemented, the development of a prototype usually is a cost-intensive and time-consuming process. An alternative to prototyping is preliminary analysis, such as analytical models, simulation, or other performance estimations, which involves reduced overhead and costs by modelling the respective systems.

Nevertheless, large quantities of design point evaluations pose a significant challenge during the design process. Addressing the increased workload in a design evaluation workflow could be accomplished through *scalability*, enabling the DSE workflow to distribute the evaluation workload and address the complexity of dCPS to achieve a time-constrained and efficient exploration of the design space. A prior literature study investigated different scalability methodologies applicable to the evaluation of dCPS designs [30]. Building on the literature study, this research considers the entirety of the evaluation environment to provide a scalable and efficient workflow for the Design Space Exploration of complex dCPS.

### 1.1 Problem Statement

Evaluating an extensive collection of design configurations, based on the available options of design choices, poses a significant challenge to researchers and designers of the next-generation complex dCPS. The evaluation of a vast volume of design points necessitates scalability in the evaluation workflow, which can facilitate and aid the design process for researchers and designers of complex dCPS. However, technology for efficient and scalable design space exploration of dCPS is a largely unexplored research field [24], where scalability of the evaluation environment is a segment of the research and development required

## 1.2 Objectives and Contributions

---

to facilitate the design process of next-generation dCPS. In order to address the challenge of scalability in the evaluation workflow, a main research question has been formulated as follows:

*How can a distributed evaluation workflow be designed and dynamically adapted to enable efficient and scalable evaluation of the vast design space of complex, distributed Cyber-Physical Systems?* (RQ1)

By extension of the main research question, the following questions will also need to be addressed:

*What are the key design considerations for developing a distributed evaluation workflow for efficient and scalable evaluation of complex, distributed Cyber-Physical Systems?* (RQ1.1)

*How can a distributed evaluation workflow be composed to scale evaluations efficiently across multiple computing resources?* (RQ1.2)

## 1.2 Objectives and Contributions

In this research, the challenge of scalability in the evaluation environment of the design process employed by researchers and designers of the next-generation dCPS is addressed. The objective is to investigate and develop a scalable and efficient approach to evaluate a vast number of complex dCPS design points by exploiting distributed simulation techniques. Achieving this objective involves the following main contributions of this research: (i) Focused literature review, where key design considerations for distributed simulation workflows are identified and the scalability challenge of evaluations in the design space exploration of complex, distributed cyber-physical systems is investigated. (ii) Conceptual development of the distributed simulation workflow, where a methodology is proposed based on the state-of-the-art and the scalability requirement of the design process. (iii) Development of a proof of concept, where the conceptual workflow design is advanced to a working implementation that provides scalability to the evaluation environment. (iv) Performance analysis and demonstration of the proposed solution and its applicability through a set of case studies employing a range of dCPS models. (v) Identification of future research directions and potential applications.

## 1. INTRODUCTION

---

The novelty and benefits provided by this research to the research field are as follows:

- (i) Advancement of knowledge in the field of design space exploration of complex, dCPS by providing a new approach to address the scalability challenge and enabling the exploration of larger design spaces.
- (ii) Practical contributions to the development of distributed simulation workflows that can be applied to a wide range of cyber-physical systems and optimisation problems.
- (iii) Experiments for an evaluation environment in the context of design space exploration for complex dCPS through a set of case studies using a variety of system models.

### 1.3 Structure

The research presented in this thesis is structured in the following manner:

#### **Chapter 2** Background and Related Works

The background and related works introduce and establish the foundational knowledge of the research area. After describing and defining the background on CPS and design space exploration, scalability, simulation and distributed computing are introduced to establish the context of the research. Following is the related works, discussing the current state-of-the-art and other relevant research in the domain of scalability for the evaluation environment in design space exploration of dCPS.

#### **Chapter 3** Approach

The approach establishes the frame, expectations, and requirements of the required solution to address the scalability challenge of the evaluation environment in the design space exploration of dCPS. Discussing the choice of simulation and simulator for the evaluation environment and the scalability capabilities already available. Additionally, the facilities, capabilities, and expectations of a distributed computing environment to enable scalability are introduced.

#### **Chapter 4** Methodology

The methodology introduces the proposed conceptual design and corresponding implementation of a scalable evaluation environment in the context of design space exploration of dCPS. Establishing a conceptual design based on the state-of-the-art and the expectations discussed in the approach section. From



the design, an implementation is built and discussed. Covering the key components and design choices that create and shape the proposed scalable evaluation workflow.

### **Chapter 5** Evaluation

The evaluation covers the performance analysis and demonstration of the proposed solution and its applicability to design space exploration of dCPS. A variety of dCPS models are employed to perform case studies investigating the performance, utility, and behaviour of the solution. Multiple experiment definitions are presented, each defining the purpose, goal, environment, metrics, and other essential components involved in each individual evaluation experiment. Concluding the evaluation are the results, which exhibit the experiment outcomes and highlight valuable aspects of the data.

### **Chapter 6** Discussion

The discussion extends on the evaluation chapter by analysing and interpreting the outcomes of the experiments. Providing valuable insights into the behaviour, performance, effectiveness, and applicability of the proposed solution under various circumstances and use cases. Additionally, a reflection on the context of the research and its research questions is presented based on the evaluation outcome and methodology.

### **Chapter 7** Conclusion

The conclusion reflects on the research and its objectives. Summarising the research process, highlighting evaluation results, and introducing the research conclusions. Lastly, recommendations for future work are presented.



## 2

# Background and Related Works

In order to address the challenges in the design of *Cyber-Physical Systems* (CPS), the basic concepts, properties, and research areas will be introduced. Following is *Design Space Exploration* (DSE), an approach to aid a designer of CPS. There the DSE2.0 project, on the challenges of advancing DSE to *distributed Cyber-Physical Systems* (dCPS), is highlighted, as it is the context in which this research is performed. After DSE, the concept and definition of scalability are examined, which is central to the aim of the research in this thesis. A brief introduction to simulation and well-known simulators is made next. To finish the background, a brief overview of *Distributed Computing*, a common methodology to facilitate scalability, is provided. After the background sections, concluding this chapter, related work on the state-of-the-art methods and techniques involved in the scalability of system-level simulation environments applicable to DSE of dCPS is presented.

## 2.1 Cyber-Physical Systems

Nowadays, Cyber-Physical Systems (CPS) are omnipresent in daily life and they drive critical information-technology infrastructure in our society. Deeply intertwining physical and software components, CPS is a broad denominator and comprises many different kinds of systems. Nevertheless, they can generally be described as (automated) systems that integrate operations of physical systems with computing and communication infrastructure [4] [26] [36]. Research and innovation is supported by these systems in a broad range of crucial industry sectors, such as robotics, industrial automation, avionics, space, and health industries. Examples of such systems can range from a simple electronic thermostat to advanced lithography machines, aeroplanes, autonomous cars, and medical monitoring

## 2. BACKGROUND AND RELATED WORKS

---

devices. Designing CPS involves many engineering principles like electrical engineering, computer science, material science, and more.

Formulating a definition is not straightforward, as a large variety of systems and engineering fields are involved. Liu et al. [37] describe CPS as multidisciplinary systems which conduct feedback control on distributed embedded computing systems. Others also include feedback control which affects the physical process and computations [35] [36] [4]. Another integral component is the communication infrastructure connecting subsystems, enabling the integration of physical processes and computation [35]. By integrating several different subprocesses and physical systems, the CPS can provide its service. The physical processes themselves are often monitored by sensors and controlled by actuators, which are orchestrated through embedded computers [36]. As many different physical components can be present, which in turn can or need to interact with other systems, CPS are a common environment for specialised hardware and purpose-built software to manage all systems.

Nevertheless, CPS have various defining characteristics, which can be described as follows [54] [37]: (i) (Complex) physical system with cyber capability. Physical systems are integrated through the software embedded in the physical component or embedded system, where control and feedback are majorly constrained by the physical phenomena and time. (Smaller-scale) CPS usually are resource (i.e., computing and network bandwidth) constrained. (ii) Networked information system. A CPS consists of a collection of subsystems which can gather, process, and communicate information. (iii) Closely integrated heterogeneous systems. CPS are built on heterogeneous distributed information and physical systems, which rely on communication and are integrated with each other. Between subsystems the system has to manage complications like time synchronisation and spatial location of (moving) physical components. (iv) Automation, Control, and Adaptivity. Components and subsystems are often highly automated and governed by control loops. As physical components are present, the system has to be robust and adaptive to address a changing and dynamic environment. (v) Security, Real-Time, and Reliability. As CPS drive critical systems, cyber-security is of great importance. There can be requirements mandating data processing or operating in real-time. Further, reliability can be necessary in the form of fail-safes or performance guarantees.

As stated before, CPS is a broad term encapsulating a large variety of systems. When considering more advanced systems from sectors like health industries, industrial automation, avionics and space, such environments are often based on multiple distributed CPS connected through an extended communication infrastructure to create *distributed Cyber-Physical Systems* (dCPS) [2]. The subsystems each implement a variety of functionalities

(e.g., monitoring and control) to provide specific services at the system-level. Advanced dCPS from manufacturing companies, such as ASML, Canon Production Printing, and Philips, are typically considered complex dCPS. A complex dCPS is a large collection of interconnected subsystems where multiple dependent compute nodes are responsible for various tasks, such as data processing, monitoring, and control. Collectively, the subsystems provide a broad range of services and features.

CPS is an active field of research where various challenges are addressed in areas such as system architecture, system autonomy, model-based development, information processing, resource management, and security [54] [37]. Relevant to this thesis is the challenge of model-based development, where the capabilities of existing methodologies are inadequate to address the complexity and scale of dCPS [24] [54]. The complex interaction between the various heterogeneous subsystems of a CPS can provide considerable challenges during the design and in the field. Important areas of concern to the operation of CPS include energy control, secure control, transmission and management, model-based software design, control technique, and system resource allocation [54].

## 2.2 DSE

Design Space Exploration is one of the approaches designers can utilise to provide themselves with early directions in the design process [24]. By aiding the designer in the early phases, DSE can accelerate the design process. Involved in the design process are the *design choices*, which determine the characteristics and performance of the system (i.e., clustered communication network or mesh communication network, general purpose CPUs or hardware accelerators, many small subsystems or merge into a few subsystems). A system can have many design choices, where each combination of design choices represents a single design solution. All design solutions together define the solution space of a set of design choices. DSE is the process of traversing the potential design solutions (*design points*) within the solution space. Generally, it aims to find an exact or reasonable solution satisfying a set of *design objectives* and typically avoids exploring the entire design space. Instead, it is time-constrained and tries to intelligently search the design space by taking into account previous evaluations of design points. Design objectives, which describe the functional and non-functional requirements of a system, are utilised by the DSE to explore, discover, and evaluate solutions. Functional requirements define a set of system features or functionalities that determine the operational behaviour of a

## 2. BACKGROUND AND RELATED WORKS

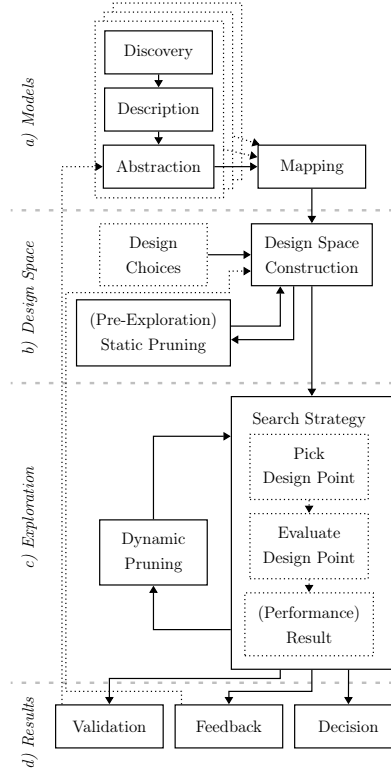
---

system. Restrictions or constraints on system attributes concerning areas such as security, reliability, performance, maintainability, scalability, and usability are captured by the non-functional requirements. Examples of commonly used design objectives include energy consumption, cost, and throughput. The goal of DSE is to discover solutions that best satisfy the design objectives as defined by the designer. The complexity and size of the design space is based on the number of design choices, classifying DSE as either a single or multi-objective optimisation problem. Additionally, design constraints may be applicable to capture limitations and requirements that need to be enforced on design points. DSE can be applied to various different scientific fields, and has already been adopted within the computer systems field in areas such as low-level hardware design for Systems-on-a-Chip (SoC) [47] and Multiprocessor System-on-a-Chip (MPSoC) [31], or the co-design of hardware and software [61]

Even though DSE has been applied in the computer systems field, the application to complex distributed Cyber-Physical Systems is a relatively unexplored area [52]. The adoption and application of DSE for dCPS is being explored by an ongoing research project "DSE2.0", which is a collaboration of the University of Amsterdam, Leiden University, and ASML [24]. The DSE2.0 project investigates the entirety of the state-of-the-art DSE process to advance from, e.g. embedded systems to large industrial machines like the ASML TwinScan machines. As existing state-of-the-art research is used as a baseline, their research is an evolutionary approach instead of a revolutionary approach. Based on their research into the state-of-the-art, they have created a general DSE workflow, as illustrated in Figure 2.1, where four main stages are outlined within the DSE framework: (i) Models, (ii) Design Space, (iii) Exploration, and (iv) Results.

In order to achieve the respective objectives based on the underlying system and design choices to be explored, it may be necessary to reconsider the implementation of individual components as it is a general workflow. An exact implementation of the DSE process can depend on diverse factors, such as the system/use-case itself, restrictions, design objectives, and available resources.

The first stage, *Modelling*, consists of four components: (i) Discovery, (ii) Description, (iii) Abstraction, and (iv) Mapping. DSE starts with the discovery phase, which explores the system and its artefacts (i.e., software, design choices, and available architectural platforms). The goal is to capture the structure of the system and the behaviour of the internal processes. Together, the system structure and artefacts provide a descriptor of the system. However, various levels of detail and complexity can be deployed in the system description. A high complexity and too detailed description are not well suited



**Figure 2.1:** General DSE workflow by Herget et al. [24].

for time-constrained and efficient analysis needed by DSE. After the description phase, an abstraction phase is present to address the complexity and detail of the initial system and artefact models. The abstraction level describes the level of detail and complexity in the initial models. In the mapping phase, the models of the system and artefacts are combined, which creates a single abstract system representation. Combining software and hardware models of the system, the abstract system representation is utilised by further steps in the DSE workflow as a baseline model.

The second stage of the general workflow utilises this initial model to build a *Design Space* (DS). By defining a set of design choices (or a set of design points and the corresponding parameters), a DS of possible design points is spanned. At first, the DS will encapsulate all possible design points based on the design choices and the initial model. Nonetheless, a (pre-exploration) pruning phase can be present in the second stage, which performs preliminary pruning on the DS based on factors such as external constraints or incompatibilities (i.e., two hardware components are incompatible with each other or a set of internal processes have specific hardware requirements).

In the third stage, *Exploration*, the previously defined DS is explored and evaluated,

## 2. BACKGROUND AND RELATED WORKS

---

capturing the characteristics and performance of the various design points available. Two major parts of the third stage are the search algorithm and the evaluation environment. The search algorithm determines how the solution space is traversed and which design points should be evaluated. As shown in the general workflow, the search algorithm can also make use of intermediate results to perform dynamic pruning. Thereby guiding or adapting the current search strategy and potentially decreasing the DS to be explored. A variety of search algorithms exist, such as exact and heuristic-based algorithms [47], each having its own approach, characteristics, and merits.

The evaluation environment is used to evaluate all design points selected by the search algorithm. Herget et al. [24] define the evaluation of a design point as the study of its extra-functional behaviour and the capture of measurable Key Performance Indicators (KPIs). There are various ways of accomplishing the evaluation of a design point, which can roughly be separated into three methodologies [47]: (i) benchmark measurements on (prototype) implementations, (ii) simulation-based evaluations, and (iii) analytical model estimations. Prototype evaluation will provide the highest accuracy of the methodologies. However, the cost and development time involved in creating prototypes (especially for dCPS) prevents the evaluation of many design points. Generally, analytical model estimations are considered to be the fastest method for evaluation. However, typically those are unable to sufficiently capture system behaviour and characteristics [47]. Simulation-based evaluations bridge the gap between prototype evaluation and analytical model estimation. Simulations can provide both high accuracy (at the cost of higher evaluation time) and fast evaluation (at the cost of accuracy). This trade-off between accuracy and speed is often determined by the abstraction level of the simulation, as mentioned by the abstraction component in the first stage (*Models*) of the general workflow. The flexibility in evaluation accuracy and speed is why many existing DSE approaches utilise simulation. The proposed approach to DSE for dCPS in the DSE2.0 project [24] also included simulation.

During the evaluation of a design point, the KPIs and other metrics of a system's behaviour and performance are captured. As mentioned before, the search algorithm can utilise the intermediate evaluation results to guide its exploration through the solution space with a dynamic pruning phase. Additionally, intermediate results can provide guidance or recommendations to the designer for design choices, whereby the designer might interject further constraints in the design process.

The final stage of the general workflow, depicted in Figure 2.1, is the *Results* stage. Both the (intermediate) outcomes and conclusive recommendations are encapsulated in this stage. Utilising the intermediate data, the search algorithm and models used throughout



the DSE workflow can be validated, tuned, adjusted, or changed. An example of such an adjustment could be in the abstraction level, where a multi-level hierarchical search approach tunes the abstraction level of the design points (i.e., increasing or decreasing the abstraction level in a set of runs). At the end of the workflow, the designer is presented with the best found design decisions to guide their decision-making in the design process [24].

Nevertheless, as this is a general workflow, the size and complexity of dCPS give rise to various challenges in the DSE process. For their position paper of the DSE2.0 project in 2022, the authors identified two research challenges in specific [24]: (i) Modelling complex dCPS, and (ii) Scalable DSE.

The challenge of modelling complex dCPS concerns specifically the first stage, *Models*. Modelling the heterogeneous subsystems of a dCPS poses a significant challenge in comparison to systems like SoC and MPSoC [24]. Where manual creation could be possible for smaller, simpler, and more elementary systems, it is deemed infeasible for dCPS. In their position paper, Herget et al. [24] propose (semi-)automatic model inference of the application and platform model as a potential viable solution.

Besides modelling complex dCPS, scalability of DSE for dCPS was also identified as a research challenge. Both the heterogeneity and complexity of dCPS increase the number of design choices present, which in turn significantly expands the DS. This inflation is further exacerbated by conditions such as dynamic behaviour of the system and its workload settings. Herget et al. [24] argue that an efficient approach to DSE is required, which can utilise the scalability of the search and pruning strategy, of the simulation environment, or both. Thereby covering the entire third stage, see Figure 2.1, with the second research challenge. Especially the Evaluation Design Point and (Performance) Result steps will be of interest, as the scalability of the simulation environment is the focus of this thesis.

## 2.3 Scalability

The second scientific challenge described by Herget et al., scalable design space exploration, concerns the need for adequately scalable new search and pruning strategies, and efficient design point evaluation. Focusing on the scalability of the simulation environment requires a grounded understanding of the scalability concept itself. Even though the term scalability is widely adapted and applied in historic and modern computer science, a single detailed definition has not been agreed upon [40] [25] [34].

## 2. BACKGROUND AND RELATED WORKS

---

A rather common colloquial usage or definition of the term "scalability" concerns the ability of a system, network, or process to be able to cope with and address an increase in the amount of work, or the potential to grow its own capacity in order to accommodate the increased workload. However, this is a very generalised interpretation of scalability. When considering scalability in the context of simulations, a classification of the *capabilities* can be expressed as follows: (i) the ability to cope with an increasing number of simulation entities (Size capability), (ii) the ability to reduce the time required to complete a single simulation run (Time capability), and (iii) the ability to facilitate more extensive and complex simulations (Complexity capability) [34].

When addressing the size capability in a simulation environment, the focus is often on throughput of simulation entities. The goal is to increase the number of completed simulation entities per time unit. The size capability goal is most commonly implemented by increasing the number of available computing resources on which simulations can be executed. Simply put, enabling more simulation entities to run simultaneously results in increased throughput.

Instead of focusing on throughput, by addressing the time capability in a simulation environment, the latency of a single simulation entity is targeted. Here the goal is to decrease the time required to complete a single simulation entity. The time capability goal is most commonly implemented through performance improvement of the hardware, parallelising a single simulation over multiple compute resources, performance improvement of the software the simulation is being executed on, or a combination thereof. Simply put, the faster a single simulation entity is running, the lower the latency.

Addressing the complexity capability in a simulation environment can concern various areas. An increased complexity capability could, for instance, allow larger or more complex models to be simulated. Additionally, it might provide the simulation environment with additional simulation capabilities or features not supported before. Compared to size and time capability, the goal and ways of achieving complexity capability are more challenging to define as there could be many ways to improve the complexity capability.

A well-known theorem related to scalability in the field of computer science is Amdahl's law [3], which formulates the expected theoretical speedup in execution time latency of a task with a fixed workload, when a system has its resources improved. Equation (2.1) demonstrates Amdahl's law, where  $s$  is the speedup of the part from the task which benefits from the improved resources (i.e., 4 when the number of resources has quadrupled), and  $p$  is the proportion of the task which benefits from the improved resources. Amdahl's law states that the performance improvement of a single part in a system is limited by the

fraction of total execution time it requires. For example, a sequential workload of 4 hours has a part of 1 hour which can be parallelised. Even in the case of infinite parallelism, the minimum execution time of the workload will be 3 hours. As a result, the theoretical speedup for a system is limited by the part of the system that does not benefit from the improved performance.

$$S_{\text{amdahl}}(s) = \frac{1}{(1-p) + \frac{p}{s}} \quad (2.1)$$

An important factor in Amdahl's law is the fixed workload/problem size (strong scaling). However, in practice, when more compute resources are available or resources improve, the problem size is often also increased to fully exploit the available computing power (weak scaling). The proportion of the parallelisable part in the total execution time grows and often more significantly than the sequential part. Another well-known theorem, Gustafson's law [23], takes the increase of workload into account and therewith provides a less pessimistic and more pragmatic view of the theoretically achievable speedup. Equation (2.2) demonstrates Gustafson's law, where  $s$  is the proportion of execution time which is spent on the sequential part of the workload,  $p$  is the proportion of execution time which is spent on the parallel part of the workload, and  $N$  is the number of processors present in a parallel computing resource.

$$\begin{aligned} S_{\text{gustafson}}(s) &= s + p \cdot N \\ &= s + (1-s) \cdot N \\ &= N + (1-N) \cdot s \end{aligned} \quad (2.2)$$

Both Amdahl's law and Gustafson's law approach scalability of a workload from different perspectives. Whilst Amdahl's law assumes a fixed workload size, Gustafson's law assumes a tendency to increase the workload size when the compute resources improve. Amdahl's law aligns with the previously defined Time Capability as both assume a fixed workload. On the other hand, Gustafson's law aligns with the Size Capability, as both assume an increase in workload size. Connecting Complexity Capability with either law is not straightforward, as both do not address the change to a more extensive or complex workload.

Where Amdahl's and Gustafson's law approach scalability from the workload size, scalability can also be viewed from a hardware perspective. A common analogy in the field of computer science is the comparison of vertical scaling and horizontal scaling. Vertical scaling concerns the improvement of a single resource (i.e., higher core count CPU or increase

## 2. BACKGROUND AND RELATED WORKS

---

in RAM size). Horizontal scaling concerns an increase in the number of resources. An integral difference between vertical and horizontal scaling is in the workload distribution. With vertical scaling, the logic of the application remains the same. However, horizontal scaling requires the workload logic to be split into smaller chunks that can be executed in a distributed fashion on multiple resources. The same comparison with scalability capabilities can be made as was done for Amdahl’s and Gustafson’s law, where horizontal scaling aligns with Size Capability, and vertical scaling aligns with Time Capability. Both could support Complexity Capability, but this behaviour highly depends on the workload and its requirements.

A simulation environment can address and implement scalability capabilities in a variety of ways. Simulation environments attempt to mimic a given system, allowing for the opportunity to assess its characteristics and evaluate its performance based on the defined design objectives. As mentioned in Section 2.2, an efficient, time-constrained analysis is needed by the DSE process. One way of achieving this is to adjust the degree of granularity (abstraction) of the model that will be simulated. A high level of abstraction will provide an increased evaluation speed (Time capability), but will have a loss in details of the system (Complexity capability), often resulting in a lower simulation accuracy [24]. By adjusting the level of abstraction, different simulation frameworks and techniques accommodate for their intended application. One of the highest granularity levels in system simulation is register-transfer level, where a system’s digital signals between registers and combinational logic are modelled. Other techniques with higher levels of abstraction include, amongst others, cycle-accurate [51] [10], transaction-level [11], and trace-driven [53].

Besides scalability through abstraction, there are various other ways of achieving scalability in simulation environments. In a previous literature study [30], the following three methodologies were identified: (i) Simulation Campaigns, (ii) Parallel Discrete-Event Simulation (PDES), and (iii) Hardware Accelerators.

Between the three approaches, a significant difference in philosophy and capabilities is visible. Hence, each method will be applicable to different system architectures. Simulation campaigns distribute the work amongst multiple resources, providing size capability. PDES splits a single model and executes the different parts on separate resources, providing time capability and complexity capability. Hardware accelerators can provide scalability through specialised hardware. As the capabilities of hardware accelerators and their integration into a system are highly specialised, the literature study remained equivocal to their adoption into the simulation environment of a DSE workflow. Composite approaches were also briefly highlighted, as they could provide a middle ground between the three

approaches. The literature study concludes that the applicability of the methodologies highly depends on factors such as the use case, platform, and goal.

## 2.4 Simulation

Section 2.2 discussed the various options that could be utilised to evaluate the characteristics and performance of design points. In the context of the DSE2.0 project, the choice was made to simulate the design points, as it can provide a (adaptable) balance between evaluation speed and accuracy. Nevertheless, there is a wide variety of simulation frameworks, which all boast their own characteristics, features, advantages, and disadvantages. Characteristics of a simulation framework can be defined by a simulation model, which encapsulates multiple dimensions such as time and state. The choice of implementation within those dimensions for the simulation model will heavily depend on the system to be simulated and the goal of the simulation.

An important part of the simulation model is the representation of time, which can generally be classified into two categories: continuous time and discrete time. The distinction between continuous and discrete is based on the interval of state transitions in the model. In a continuous time simulation, state transitions occur continuously throughout time. Whereas in a discrete time simulation, state transitions occur in discrete time intervals. An example to distinguish the two can be made with a sine wave, where the x-axis is time. Continuous time simulation will encapsulate the entirety of the sine wave and will have a value for all timestamps. However, discrete time might only have the values for the timestamps where the sine wave reaches its extremes (-1 and 1). All values between the extremes of the sine wave are not captured.

The discrete time simulation can subsequently be divided into two general categories: time-driven and event-driven. A time-driven discrete simulation indicates that state transitions occur at discrete time intervals (commonly referred to as time steps). Whereas event-driven discrete simulation (more commonly known as *Discrete-Event Simulation*) advances in time based on event activities (i.e., process x sends a message to process y).

Discrete-Event Simulation is commonly utilised in the field of computer system simulations, like SoC and MPSoC systems. The choice for a discrete time model seems like a reasonable fit, as a computer system processes data as discrete values (values are processed as 0s and 1s) and state of the system changes at discrete time intervals (i.e., based on the clock frequency). Instead of the time-driven approach (i.e., based on the clock frequency),

## 2. BACKGROUND AND RELATED WORKS

---

an event-driven approach (i.e., a computation is starting or a file read has completed) can also be used.

When simulating an entire system (system-level simulation), the behaviour of system components and processes have to be captured. Various different solutions exist to model the behaviour of such a system, like trace-driven and program execution-driven [50] [49]. A trace-driven approach utilises traces of program execution and system component states to reconstruct and mimic the behaviour of the system. As traces of program execution are utilised, an existing system is often used as baseline. Program execution-driven is similar to the trace-driven approach. However, the program execution-driven approach has to read a program and simulate the execution in the system on the fly. Program execution-driven simulation requires a program file, which is usually orders of magnitudes smaller than the trace file(s) required for trace-driven simulation. However, the execution-driven approach is cost-intensive and detailed as the simulation has to process instructions one-by-one and update the system state accordingly. This is further exacerbated when a large number of simulations have to be performed. Hence, Herget et al. [24] deem the trace-driven approach a better fit in the context of DSE2.0.

A broad variety of well-known simulators exist and have been used to research the design and performance of a diverse set of computer systems (i.e., from SoCs to large-scale networks). Such simulators include OMNeT++, gem5, SystemC, NS-3, and OPNET, where each simulator will have its own unique feature set and intended purpose. The gem5 simulator is a computer-system architecture platform supported and utilised by academia and industry, such as the National Science Foundation, AMD, ARM, Hewlett-Packard, IBM, Intel, Metempsy, Micron, MIPS, Samsung, and Sun [22] [58]. Governance is arranged as a meritocratic, consensus-based community project. Gem5 is a discrete-event simulator providing simulation both at system-level architecture and processor microarchitecture.

Similarly to gem5, OMNeT++ is also a well-known open-source discrete-event simulator, which has gained widespread popularity as a network simulation platform both in the scientific community and in an industrial setting [59]. Simulation models in OMNeT++ are component-based, where modules are programmed in C++ and assembled in larger components/systems using a high-level language (NED). The modularity provided through the components structure and NED system provides reusability of models. Additionally, both an extensive GUI and command-line interface are provided. Due to the modular structure of the simulation kernel and the project being open-source, extensions can be made to the simulator (as shown by the large variety of community projects adapting

or extending OMNeT++, like the INET project, which provides tools for research on communication networks).

## 2.5 Distributed Computing

Section 2.3 discussed the concept of scalability, where it was established different ways exist to achieve scalability. Nevertheless, a methodology commonly used to achieve scalability is horizontal scaling, which distributes the workload over multiple (interconnected) resources. The process of distributing a workload over multiple resources concerns the domain of *distributed computing*. Distributed computing makes use of a network of computing platforms (a distributed system) to facilitate computing, information access, and information exchange in order to process a workload [33] [32]. A wide variety of research fields, such as physics, medical sciences, and avionics, utilise distributed computing to compute on intensive workloads, like large-scale space simulations [27], protein folding [8], and CFD aerodynamics simulations [55].

Distributed computing and distributed systems are considerably different from computing on traditional single-machine systems. Distributed computing allows the application to leverage multiple resources to reach a common goal. Compared to a single system, distributed systems are significantly more complex to utilise effectively. Additionally, effective utilisation also depends on the characteristics of the application. Common characteristics and challenges can be identified for distributed computing and utilising a distributed system: (i) communication, inherently the subsystems of distributed computing require a communication mechanism to establish interaction. A common methodology for communication is message passing, where a network is used to enable each process to send and receive messages. The Message Passing Interface (MPI) standard is widely known and utilised [60]. (ii) synchronisation, in order to keep consistency, as independent subsystems interact, a synchronisation methodology is required. When multiple independent processes can execute concurrently, the interaction between them has to be regulated. Otherwise, the system will lose consistency of its state. The implementation of a synchronisation protocol is application specific as it depends on factors and characteristics such as strictness, communication, and architecture. (iii) data management, distributed computing is often used in applications which process and/or generate large quantities of data. Storing, moving, and communicating large quantities of data can be very costly and an inherent challenge of an application. Smart placement and interaction with data could enable improved use of the capabilities in a distributed environment. Lastly (iv) utilisation, where distributed

## 2. BACKGROUND AND RELATED WORKS

---

computing, usually, makes use of multiple compute resources, a common goal is to utilise those resources as effectively and efficiently as possible. Utilisation can be considered at different scales, from a single resource to entire compute clusters. Energy efficiency has become a prevalent area of research and often considers methodologies to improve utilisation or reduce energy consumption.

One of the applications of distributed computing is scientific computing, where systems like cluster computing, grid computing, and cloud computing are utilised and researched. Compute clusters are a large collection of (often homogeneous) compute resources, which are usually orchestrated to cooperate on a task. When a compute cluster is shared amongst users, Cluster Management Software (CMS) can be used to supervise a job-batching system. By allowing users to submit jobs to a central queue, the CMS can ensure cluster-level goals such as fairness and utilisation. An example of a cluster computing initiative is the Distributed ASCI Supercomputer (DAS) by the Advanced School for Computing and Imaging (ASCI), which involves Dutch universities and research institutes [5]. DAS has multiple cluster sites located at universities and research institutes, allowing researchers and students to perform research on compute clusters.

Grid computing is related to cluster computing but has some significant differences. Where cluster computing generally is a local area network of homogeneous compute resources, grid computing is a widely (geographically) distributed network of primarily heterogeneous compute resources. Grid computing can utilise heterogeneous generic computing resources (i.e., personal computers or workstations) to create a distributed computing network. An example of a grid platform is Grid5000 in France [6]. Grid5000 aims to provide resources for experiment-driven research in the field of computer science, with a focus on parallel and distributed computing, including Cloud, High-Performance Computing (HPC), Big Data and AI. Another well-known application utilising a grid infrastructure is the Folding@home project, which is a distributed computing project aimed to aid scientists by simulating protein dynamics. It provided biological insights for drug discovery and other efforts to combat global health threats [8].

Lastly, the increasingly more popular cloud computing platform, which targets on-demand availability of computer system resources (i.e., data storage and compute power) to external customers. Cloud computing has evolved from grid computing to a service-oriented structure boasting abstracted, virtualised, dynamically scalable, managed computing power, storage, platforms, and services on demand [21]. Many large companies in the tech sector, like Amazon, Google, and Microsoft, have created their own cloud infrastructure for external customers.



## 2.6 Related Works

In order to research scalability of system-level simulation environments applicable to DSE of dCPS, the state-of-the-art on methods and techniques involved will need to be explored. The related works section is based on and extends on the research presented in a previously published literature study on the scalability in system-level simulation environments for dCPS [30], where scalability and available methodologies were investigated.

### 2.6.1 Scaling Frameworks

Firstly, the capabilities of simulation frameworks (like OMNeT++) and other toolchains will be investigated. Usually, a distinction can be made between the following two cases for scaling tools [7]: (i) scalability built-in, where simulation frameworks have scalability tools built in, or (ii) add-on scalability, where external applications extend the scalability facilities of an existing simulation framework.

#### 2.6.1.1 Simulator Capabilities

The well-known discrete-event simulator OMNeT++ has several capabilities to provide small-scale scalability. A basic sequential simulation, which runs on a single CPU, is provided out-of-the-box. However, it features tools to facilitate improved scalability through simulation campaigns and parallel discrete-event simulation [59]. More specifically, OMNeT++ provides the tools to perform a *parameter study* on a model, which is used to explore the parameter space, or to perform repetitions using different seeds for a random number generator to increase the statistical accuracy. The parameter study can be performed on a single CPU or multiple CPUs in the same computing system. A drawback of the available tool is that it only supports a single configuration of the model, classifying it as Multiple Replications In Parallel (MRIP). Parallel execution of different model configurations at the same time is not supported. Nevertheless, only a single processor will not be sufficient to complete large-scale scientific simulations or exploration of a vast design space in a manageable time frame. OMNeT++ refers to batch-queuing, cluster computing or grid computing middleware for the adoption of (distributed) compute clusters [59]. The parallel discrete-event simulation support of OMNeT++ allows the developer to indicate separate subprocesses within a model. Those subprocesses will then be executed independently from each other. However, the parallel discrete-event simulation setup is a manual process with quite some constraints and limitations [59].

## 2. BACKGROUND AND RELATED WORKS

---

Another well-known community-led simulator is `gem5`, which is intended as a platform for computer-system architecture research, both at a system-level as well as processor microarchitecture. Similarly to OMNeT++, the standard is a basic sequential single CPU simulation. However, a big community is supporting the development of `gem5` and creating projects of their own. One of the extensions made to `gem5` is the `dist-gem5` project [44] [38]. `Dist-gem5` provides support for distributed system modelling, which splits large distributed models into separate processes (like Parallel Discrete-Event Simulation) that can be executed on separate resources. Communication and synchronisation between the separate instances is facilitated by `dist-gem5` through TCP connections and quantum-based synchronisation.

### 2.6.1.2 Scalability Toolchains

Besides the built-in scalability features of simulation frameworks, other research or community projects also try to provide scalability in a simulation environment. One of those projects is Akaroa [19], which is actually available in OMNeT++ [59]. Akaroa aims to speed up the simulation process for quantitative stochastic simulation using the Multiple Replications In Parallel (MRIP) paradigm, which is a form of simulation campaign. A central process accumulates the data of the replications and determines, based on a predefined confidence/precision level, whether more observations are required. Akaroa concludes the simulation campaign when enough observations have been accumulated.

A project that aims to provide scalability in managing simulation campaigns and post-simulation analysis for OMNeT++ is SMO (Simulations Manager for OMNeT++) [7]. SMO is similar to the parameter study built into OMNeT++, as it analyses the configuration file of a model and will execute all parameter value combinations. As with OMNeT++, it is limited to the computing resources of a single machine. However, their focus is on the management and post-simulation analysis of data. SMO has separate summariser and analysis tools that process the output of the parameter study, calculate statistics, and provide (interactive) visualisations of the results [7].

The Neurogenesis project aims to provide the ability to perform the aforementioned OMNeT++ parameter study on a compute cluster using the Message Passing Interface (MPI) [20]. Similarly to SMO, Neurogenesis generates all parameter combinations based on a configuration file. Neurogenesis makes use of a single manager which distributes all scenarios to worker instances through MPI. Whenever a simulation has completed, the worker will signal the manager accordingly.

The aforementioned scalability tools extend or are built on existing simulation frameworks. However, workflow tools like Celery [12], Dask [15], and Luigi [39] can also provide scalability even when not explicitly designed for a simulation framework. Celery, Dask, and Luigi can provide workflow management, dependency resolution, and scheduling. A task can be defined that encapsulates the entirety of a single simulation. Based on the task definition, the tools can orchestrate the set of simulations that need to be performed. Additionally, tools like Dask can distribute the workload across multiple machines or on a compute cluster through their resource managers (i.e., Slurm [63]). Such tools could provide scalability without the costly development and maintenance of a workload management and distribution tool built from scratch, allowing the developer to focus on higher-level features and services.

The scalability toolchains discussed in this section all have a similar approach to scalability, where the focus is on MRIP. Akaroa aims to speed up the simulation process for quantitative stochastic simulation using MRIP on a network of computers. The SMO project extends on the parameter study facility provided in OMNeT++ and provides management and automated post-simulation analysis of the evaluation data, which could be helpful to the designer in the DSE process. The Neurogenesis project aims to provide support for the parameter study on a compute cluster using the Message Passing Interface (MPI). However, as was the case for the parameter study tool in OMNeT++, supporting only MRIP, the utilisation of a single compute resource, or both, is not sufficient for DSE of complex dCPS. Nevertheless, MRIP could be part of DSE for dCPS, but it does not enable the search of a large-scale design space containing a broad range of system models. Instead, in this research, the aim is to address the challenge of efficient and scalable evaluation of a vast number of (independent) design points for DSE of complex dCPS, where workflow tools like Celery, Dask, and Luigi could be a valuable asset enabling the large-scale evaluation required.

### 2.6.2 DSE Frameworks

In previous sections, scalability in simulation environments has been discussed. As this thesis investigates scalability of simulation environments in the context of DSE for dCPS, this section will discuss some of the well-known DSE frameworks used in relevant research fields. Existing scalability solutions inside the DSE frameworks will be highlighted.

SESAME (Simulation of Embedded System Architectures for Multilevel Exploration) is a system-level modelling, simulation, and exploration framework for embedded MPSoC systems [48]. The framework was not explicitly designed with an aim towards scalability of

## 2. BACKGROUND AND RELATED WORKS

---

the evaluation environment. Nonetheless, the structure of the application and architecture models allows for reusability and a reduction in compilation. Additionally, the application models are executed using POSIX threads. As SESAME is focused on the smaller-scale MPSoC systems, it is not applicable or capable of facilitating reasonable performance for large-scale complex dCPS systems, as also concluded by Herget et al. [24].

Another DSE framework called NASA (Non Ad-hoc Search Algorithm) aims to create a modular general infrastructure for DSE workflows [29] [28]. Jia et al. [29] address the ad-hoc nature of software infrastructures created to facilitate the system-level DSE experiments. The NASA framework creates well-defined interfaces between the components of a DSE workflow. A modular setup allows the developer to easily integrate a variety of simulation tools and search algorithms in a plug-and-play fashion. Additionally, a dimension-oriented DSE approach is available, providing the possibility to deploy multiple search algorithms that simultaneously co-explore a design space [29]. NASA does not necessarily provide out-of-the-box scalability of the evaluation environment, where a large number of simulations can be performed. However, it does provide a modular framework where various approaches can be adapted, switched, or combined to utilise and explore their scalability capabilities.

Both scalability in the search algorithm and the evaluation environment are explored by the authors of the DeSpErate++ project [42], an extension on DeSpErate [41]. The search algorithm tries to prune suboptimal regions of the design space, thereby reducing the number of simulations required. Concerning the scalability of the evaluation environment, DeSpErate++ supports the usage of a multicore machine or a compute cluster to scale the number of simulations running in parallel. Additionally, a workload scheduler is present that employs execution time prediction to improve the utilisation of available computing resources.

When exploring literature on scalability employed by existing DSE frameworks or in DSE research, most literature encountered does not explicitly focus on the evaluation environment. A common topic is the search algorithm, where scalability can be achieved by effectively reducing the size of the design space or reducing the number of evaluations to reach an optimal or reasonable solution. An example of such a project is DISPATCH [57], where a two-step method is employed for efficient DSE of CPS, which first creates a coarse design and, in the second step, fine-tunes it to meet system requirements. The authors demonstrate the capability of DISPATCH to require fewer simulations to synthesise valid designs in comparison to various other search algorithms and human designs.

One of the recent developments in this area is project FARSI (Facebook AR system investigator) [9]. FARSI presents a DSE framework for Domain-specific SoCs (DSSoCs),

which are SoCs with domain-specialised hardware blocks. DSE for DSSoCs has to deal with complex systems due to specialised hardware, which results in high development effort. Additionally, the complexity of the design space is increased by the many specialised systems, which further complicates the search for an optimal solution. Boroujerdian et al. [9] address the complexities of DSSoCs by identifying features that are required to construct a DSE framework capable of efficiently traversing the complex design space of DSSoCs. Their evaluation of the FARSI framework shows promising results.



## 3

# Approach

In order to create a scalable evaluation workflow, the approach will review constraints, requirements, desired features, and high-level design choices. At the core of the evaluation environment are the choice of simulator and design points. Section 3.1 explores the structure of system models and available scalability in the simulator of choice. As a platform to enable scalability, distributed computing is utilised. Section 3.2 will discuss the requirements and expectations for the scalability of the distributed computing environment.

### 3.1 Simulation

The evaluation environment is used to evaluate all design points selected by the search algorithm, studying their extra-functional behaviour and capturing measurable KPIs. As discussed in Chapter 2, various methodologies exist to evaluate a design point, roughly separated into benchmark measurements on (prototype) implementations, simulation-based evaluations, and analytical model estimations. Although prototype evaluations provide the highest accuracy of the methodologies, the cost and development time prevents the evaluation of a large collection of design points. Improved evaluation speed can be offered through analytical model estimations (considered the fastest method for evaluation). However, usually, those are unable to sufficiently capture system behaviour and characteristics [47]. Instead, the simulation-based approach can provide both high accuracy (at the cost of higher evaluation time) and fast evaluation (at the cost of accuracy), bridging the gap between the prototypes and analytical model estimations. The abstraction level of the simulation often determines the trade-off between accuracy and speed. As the simulation-based approach facilitates flexibility in evaluation accuracy and speed, this research, along with many existing DSE approaches, will utilise a simulator to evaluate design points.

### 3. APPROACH

---

Central to the evaluation environment is the simulator, which will be the discrete-event simulator OMNeT++ [59]. The choice for OMNeT++ is based on the large community, active development, rich feature set, and high adaptability. Next, an overview of the structure of a design point and an analysis of the available built-in scalability tools is provided.

#### 3.1.1 Design Points

A design point needs to be created before the evaluation environment can perform an assessment. As depicted in Figure 2.1, the search algorithm is expected to provide the design points, which it will need to build according to the specifications of the simulator. System models in OMNeT++ are based on a modular component architecture, where each component (called a module in OMNeT++) is programmed in C++. Components can be assembled into larger components and models using the high-level Network Description (NED) topology description language. Between components, communication is facilitated through exchanging messages, which are defined in corresponding message files (.msg). Configuration files (.ini files) can be used to set parameters for components of the system models and to set parameters for the execution of the simulation. The modular component architecture allows for reusability between system models.

As OMNeT++ and its component architecture are used in the evaluation environment, the design points are defined through the corresponding C++, NED, msg, and ini files. The evaluation environment will not be involved in the creation of the files for a design point, nor will it be altering or expanding on the corresponding files. However, compilation of the corresponding design point files will be the responsibility of the evaluation environment. In order to facilitate a variety of different dynamic, complex interactions with external applications (i.e., a search algorithm), the evaluation workflow will need to support both continuous batch and individual inputs of design points. Different search strategies might have different requirements, and the evaluation environment aims to have a dynamic adaptation with respect to the input characteristics of the search algorithm.

In order to keep track of design points being processed by a variety of different stages, an identifier is required throughout the entirety of the evaluation environment. Retrieving results of evaluation, storing runtime data, design point caching, and communication are examples of tasks where such an identifier is required. Additionally, a designer might have the need to reconstruct the corresponding design point after the entire DSE process has been completed. In order to account for the identifier and reconstruction of a design point, the evaluation workflow will need to facilitate a token (or a token for each purpose) that



enables the respective purposes. Such a token could be generated through a hash (i.e., MD5 or SHA) or another (central) mapping process.

### 3.1.2 Scalability

In order to create a scalable evaluation environment, the capabilities already available in OMNeT++ will need to be investigated. As discussed in Chapter 2, the baseline facility provided is a basic sequential simulation running on a single CPU. Small-scale scalability facilities are included through local simulation campaigns in the form of a parameter study and Parallel Discrete-Event Simulation (PDES)[59]. A parameter study explores the parameter space or can be used to perform repetitions using different random number generator seeds to increase statistical accuracy. The study can be executed through the same simulation instance. However, the entire process would be running in sequential order and the system is prone to failure, as a failure of a single run would abort execution or corrupt the state for subsequent executions. The latter drawback can be prevented by executing every point in the parameter space in its own environment. However, this induces a setup overhead for each evaluation, and the simulations are still running sequentially on a single CPU.

In order to mitigate the aforementioned drawbacks, OMNeT++ provides the *opp\_runall* tool [59]. It facilitates the parameter study using multiple CPUs and multiple processes. The parameter set is divided into batches, where each batch will be processed sequentially in a single environment. As the batches are independent of each other, they can be scheduled and processed on multiple CPUs. The number of CPUs and batch size can be configured by the developer. A drawback of the *opp\_runall* tool is that only a single system model is supported, classifying it as Multiple Replications In Parallel (MRIP). Parallel execution of different system models is not supported.

The parameter study, using the *opp\_runall* tool, only utilises a single processor, whereas large-scale scientific simulations or exploration of a vast design space will require significantly more computing resources to complete in a manageable time frame. Large-scale simulation campaigns could utilise the large collection of compute resources from a compute cluster. However, OMNeT++ does not natively feature support to deploy on a compute cluster, instead referring to batch-queuing, cluster computing, or grid computing middleware for the adoption of (distributed) compute clusters [59]. Community-based projects, such as SimDistribution [46], SimProcTC [18], Neurogenesis [20], and OSM [7], exist, but do not fully satisfy the needs and requirements to explore a vast design space in the context of DSE for dCPS. The older (2009) SimDistribution project does allow

### 3. APPROACH

---

separate design points to be executed on different computers. However, the setup of the compute infrastructure, management and creation of the design points, and the operation of the environment is a manual process not suitable for DSE of dCPS. As discussed in the background and related works, the focus on MRIP by the Neurogenesis and OSM projects also does not satisfy the demands of DSE for dCPS.

Where the parameter study capability utilises multiple sequential execution instances, PDES alleviates the restriction of sequential simulation by splitting a single system model into multiple independent Logical Processes (LPs). The simulation is parallelised by executing the LPs on separate compute resources, where each LP will be running as an independent process. Communication between LPs is performed through message passing or file-based communication orchestrated by the OMNeT++ simulation engine.

Configuration of PDES in OMNeT++ is performed manually and does have some constraints and limitations. Dividing the system model into LPs is not automatic, but is specified in the model configuration. After the model is split into LPs, a conservative synchronisation protocol is employed, which guarantees correct order of execution, but could experience a degradation of performance (compared to the ideal speedup) or even a slowdown compared to the standard sequential execution. The modularity of OMNeT++ does allow developers to create custom communication and synchronisation classes.

## 3.2 Distributed Computing

A single sequential simulation instance running on a single compute resource will not be sufficient to facilitate exploration of the large-scale design space for dCPS in a manageable time frame. In order to support a scalable evaluation environment, distributed computing will be employed. Distributed computing allows the evaluation workflow to utilise multiple computing resources. This section will discuss the facilities, interactions, and expectations of the distributed computing environment.

### 3.2.1 Interaction

The interaction of the evaluation workflow with the designer and external application (i.e., the search algorithm) is an essential aspect of the solution. Reducing the complexity of deployment to achieve a straightforward setup process, whilst also facilitating a more advanced usage to tailor the environment to a specific use-case, needs to be achieved. Interaction with the environment will be through a central point of access, where tasks such as submitting design points for evaluation, retrieving results, blocking wait on a set

of evaluations, and a shutdown need to be available. Additionally, the configuration of the workflow and its individual components should be clearly defined, have reasonable defaults, and be validated before execution of the environment.

External applications (i.e., a search algorithm) build the configuration of the workflow and each individual design point. In order to initialise the evaluation environment, the corresponding workflow configuration is submitted. Once the evaluation environment has completed its initialisation, which includes verification and compatibility of the configuration parameters, initialising each individual workflow component, and the setup of the compute resources, the external application can interact and trigger the corresponding functional handlers. For instance, the search algorithm can instantiate a set of design points. These design points can then be passed to the central workflow manager, which processes the data accordingly. After submitting, the search algorithm can either continue operation or wait for the evaluation results of the submitted design points. Additionally, the central workflow manager can offer other handlers, such as a wait on the results of a specific set of design points, wait on all design points currently in the workflow, cancel all design points currently in the workflow, or halt the evaluation environment. After having evaluated a collection of design points, the search algorithm can access the corresponding output and local design point storage.

### 3.2.2 Facilities

Various distributed computing platforms (i.e., compute cluster, grid, or cloud) could be utilised by the scalable evaluation environment. An approach that can be adopted by multiple different platforms would provide to a variety of end-users. Nevertheless, this thesis will assume the adoption of a compute cluster, as this reduces the complexity in the orchestration of an evaluation workflow through its CMS and (often) homogeneous interconnected resources. Different scalability methodologies (i.e., simulation campaigns and PDES) can be deployed on a compute cluster, where each methodology might facilitate different DSE strategies or system models. The workflow should support both simulation campaigns and PDES to provide for a broad range of use cases. Simulation campaigns can make use of the evaluation environment to run independent processes on separate resources. PDES will utilise the facilities in OMNeT++ to orchestrate the LPs, which will be running on separate resources. The evaluation environment has to be aware of the PDES execution, as it should not oversubscribe resources. Oversubscription of resources occurs when, for example, two processes are assigned the same single resource at the same time. Depending on the computing hardware and application workload characteristics,

### 3. APPROACH

---

oversubscription could potentially lead to a degradation of system performance. The environment should not exclusively facilitate scalability through campaigns or PDES. Features already integrated into OMNeT++ should also be available to the DSE process, such as the parameter-study feature. Additionally, the simulation environment should not inhibit or restrict the configurability available within OMNeT++.

When design points arrive in the evaluation environment, they need to be processed and sent to the computing resources. Input processing also needs to be scalable in order to provide a scalable evaluation environment. One way to address the dynamic control at the input is to employ an input queue (or multiple) with a scheduling policy, where the search algorithm (or the designer) can indicate priority or a hierarchy between design points. The scheduling policy determines the order in which the design points are to be processed. However, the order of processing need not be the order of completion, as this highly depends on the evaluation execution time of the individual design points. The evaluation environment should offer multiple basic scheduling policies (like First In First Out (FIFO) or simply random order) and provide the designer with the opportunity to define additional priority-based policies (like Highest Abstraction First in the context of DSE). User-defined priority policies could be defined through metaconfiguration during the initialisation of the environment, where priority is defined through the attributes of a design point available at runtime. Besides solely scheduling the design points, the scheduling policy could also take into account the available computing resources. Advanced scheduling policies might consider the performance characteristics of heterogeneous computing resources in a distributed compute cluster or prioritise different goals/metrics, such as energy efficiency or average latency.

After the design points have been accepted at the input and processed accordingly, they can be evaluated on the compute resources. Nevertheless, the evaluation environment will need to manage the available compute resources. Different distributed computing clusters might have different CMS through which the compute resources can be reserved (i.e., a well-known interface is Slurm). Preferably, the evaluation environment should support the adoption and integration of multiple CMS. During the lifetime of the evaluation environment, it should manage the available resources, be able to scale up and down if necessary, and provide an interface for the search algorithm to submit and evaluate design points.

### 3.2.3 Data Management

As the evaluation environment is part of a larger DSE workflow, the corresponding evaluation results need to be collected and stored. The search algorithm in particular will require the results of the evaluation of each design point. As discussed in Subsection 3.1.1, the design point simulation instances will contain the necessary data collection instructions. All the evaluation results will be recorded and stored such that the other components (i.e., the search algorithm) can access them using the identifier of the design points when necessary.

Besides the results yielded from the evaluation of design points, the aim is to collect as much data as possible about the evaluation workflow. Such data includes compute resource runtime data, scheduling decisions, and general runtime execution events. By collecting as much data as possible, in-depth analysis can be performed. During or after the DSE process, the designer might want to analyse the performance or behaviour of the evaluation environment. By recording and providing all runtime data, external or internal tools may be able to provide further insights to the designer. It also allows the system maintainer to track potential erroneous results and behaviour.

Another aspect which needs to be addressed by the evaluation environment is validation. Whenever irregularities or erroneous behaviour occurs, the evaluation environment should detect and address it accordingly. A failed simulation might be addressed by retrying or notifying the search algorithm or designer of the failure. Thorough validation is required to detect inconsistencies in or between evaluation environment components, as erroneous behaviour could corrupt the state of the environment or the evaluation results. A basic validation strategy is to halt the environment whenever any irregularity is detected. However, a more balanced approach is desired, as such a hard stop is not user-friendly and could inhibit an effective design process.

As stated before, the evaluation workflow aims to collect as much data as possible. In order to facilitate extensive data collection, a data storage solution needs to be present. At first, the evaluation of a single design point will rely on storage accessible locally to the compute resource. A predetermined structured storage, which is uniform for all design points, will allow the evaluation environment to manage all data (i.e., results, runtime, and performance data) generated during an evaluation. After an evaluation has been completed, the locally stored data needs to be transferred to a central global storage. A similar strategy using a predetermined structure will need to be employed in this global storage. When all data corresponding to the design point has been transferred to global storage, the search algorithm is signalled that it can safely access the required data.



## 4

# Methodology

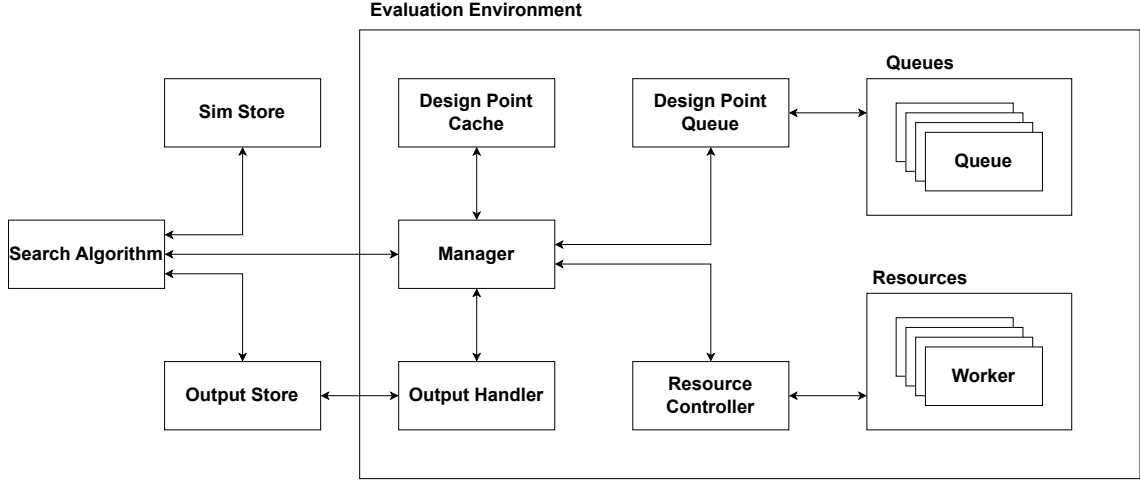
Previous sections described the concept of scalability, available methodologies, challenges, and the need for a scalable evaluation environment. In this section, a design and implementation for a scalable workflow is proposed. Building a scalable evaluation environment involves many design choices, which all affect one another and the overall environment. The design choices that have been made will be explained in detail and are majorly based on the context of the workflow and the state-of-the-art. An overall design of a scalable workflow will be presented and details on the implementation enabling scalability is provided. Challenges, observations, and findings when applying scalability in practice will be discussed.

## 4.1 Design

As this thesis is performed in context of DSE2.0 [24], the evaluation environment is assumed to be integrated into the DSE general workflow (presented in Figure 2.1). Specifically, it receives input from the search algorithm and thereby replaces the *Evaluate Design Point* and *(Performance) Result* phases from the Exploration stage. Figure 4.1 illustrates the design of the scalable workflow, which consists of five major components: (i) Manager, (ii) Design Point Cache, (iii) Design Point Queue, (iv) Resource Controller, and (v) Output Handler.

This design is constructed to address the challenges outlined in this thesis so far: (i) Dynamically Configurable Workflow, where the workflow of the evaluation environment should adapt according to the configuration of its components and the configuration of the design points provided. (ii) Dynamic Input Handling, where the workflow should support complex and dynamic interactions with external applications (i.e., a search algorithm). The

## 4. METHODOLOGY



**Figure 4.1:** Scalable evaluation workflow

design point cache and design point queue can facilitate this through scheduling policies, hierarchical queues, and comparative analysis of the design points provided. (iii) Workload-level and Task-level Parallelism, where the main methodology of each are simulation campaigns and PDES respectively. Further task-parallelism should be supported to allow for additional paradigms present in simulators or any custom approaches required. (iv) Simulator Agnostic, where the distribution and processing of the design points by the workflow is agnostic to the underlying simulator. The evaluation environment can be tailored to a specific simulator framework by specifying its corresponding evaluation routine. When evaluating a design point the corresponding approach of a simulator is performed. (v) Computing Environment Agnostic, where the computing infrastructure utilised by the evaluation environment can both be scaled from a smaller-scale (for example, a laptop during development) to a larger-scale (for example, an entire compute cluster during deployment) and utilise various CMS.

Together, these elements facilitate an efficient and scalable evaluation environment addressing the challenge of scalability in DSE for dCPS. By enabling complex, dynamic interactions and workflows that can provide diverse methodologies of scalability, adaptable to a variety of environments and infrastructures, the evaluation environment can be adopted and tailored to suit the needs, requirements, and established environments of designers and researchers of the next-generation dCPS.

The main component of the workflow is the manager, which oversees and orchestrates the entire evaluation environment. Interaction of the search algorithm with the evaluation environment is performed through the manager, where it can send design points, request



evaluation results, halt the workflow, and perform a variety of other tasks. Whenever such a request arrives, the manager will process and forward it accordingly. By creating a single point of access, the complexity of interacting with a scalable evaluation environment in the DSE workflow is reduced. Overall, the primary responsibility is to accept incoming requests, manage the workflow components, and communication between the workflow components.

Orchestration of the evaluation environment also includes recording runtime diagnostics. By tracking the behaviour and performance of the workflow, potential problems or failures can be investigated. During the execution, the manager will keep track of runtime data concerning the behaviour of the environment, such as the total number of (successful) evaluations, total runtime, total number of failures, average waiting time, and more.

Besides input requests, orchestration, and diagnostics, the manager is also where additional tasks like aggregation and validation could be performed. Aggregation refers to the task of processing metrics and data generated by an evaluation, which in turn can be utilised by the constraint checking of the search algorithm. Such aggregations and metrics could be defined through meta-configuration files crafted by the user. Validation includes the validity of the individual components in the workflow and the evaluation of a design point. The manager has to ensure that the evaluation of design points is valid and successful. When inconsistencies or violations are detected, an appropriate response (i.e., evaluation retry or hard stop of the environment) has to be taken. Although aggregation is currently not present in the evaluation environment, basic validation of design point evaluation and validation of configuration files is in place, where irregularities and exceptions will be caught and propagated to the manager, which subsequently halts the evaluation environment.

When the search algorithm provides a design point, the manager will forward it to the design point cache. Inside the cache, a store keeps track of the status (i.e., status such as processing or finished) of the design points which have passed through the environment. When the cache indicates the input is already being processed or has already been processed, the manager will not forward it to the design point queues. Instead, when the evaluation results are requested, the manager can forward the data of the previously evaluated identical simulation instance. A cache is present in the workflow to reduce unnecessary evaluations occupying valuable compute resources and to cater to a variety of different search algorithms, which may or may not implement caching themselves, thereby addressing the dynamic input handling and dynamically configurable workflow challenges.

## 4. METHODOLOGY

---

After the cache has indicated that the design point has not appeared in the evaluation environment before, the design point will be forwarded to the design point queues. To further address the dynamic input handling challenge, the workflow contains multiple design point queues to facilitate various evaluation strategies, which different DSE workflows might require. A variety of complex interactions can be facilitated, such as batch evaluations, one-by-one evaluations, or a hierarchical approach, depending on the preference and requirements of an external application (i.e., a search algorithm), thereby enabling both dynamic input handling and a dynamically configurable workflow. The workflow configuration set by the user indicates the queues present and their internal settings. The user can indicate priority between the queues, where the workflow will first evaluate design points from the highest priority queue (if not specified otherwise). When the search algorithm provides design points to the manager, it has to specify the queue identifier to which the design points belong. Each individual queue also has a scheduling policy that can be set by the user, thereby indicating a local priority between design points in the same queue. In order to support more complex scheduling policies, such as highest abstraction first in the context of DSE, the design points accepted by the environment can be annotated with metadata.

Once the search algorithm commands the environment to start evaluating, the manager will request design points from the design point queues and forward them to the resource controller. The primary responsibility of the resource controller is the execution of design point simulations by managing the compute resources and distributing the workload, which concerns the workload-level and task-level parallelism challenge. Evaluations can be performed both asynchronously (resource controller immediately continues after distributing the workload) and synchronously (resource controller explicitly waits for the simulations to complete). This allows the search algorithm to continue its own workflow whilst the design points are being evaluated in the background. When a design point arrives at the resource controller, it will be sent to one of the available worker processes running on the compute resources. In order to facilitate services such as synchronous waits on a set of design points or communicating status updates, the design points currently being evaluated are tracked. After the design point has arrived at the worker process, the source files are compiled into an executable. If the compilation is successful, the worker will continue with the simulation and capture the corresponding runtime data and evaluation results.

The last step of the evaluation environment is to retrieve the runtime and evaluation data of the design points, which is the responsibility of the output handler. Data generated by the evaluation environment is divided into three global stores: (i) (Intermediate) Results

Store, (ii) Logs Store, and (iii) Runtime Store. The (Intermediate) Results Store contains the evaluation data (i.e., KPIs) required by the search algorithm to perform DSE. All data generated by loggers, configuration, and command-line output is archived in the Logs Store. The Runtime Store captures the behaviour and performance of the evaluation environment by recording information such as average evaluation time, number of evaluations, and resource utilisation. When the output handler receives a design point, it will transfer the data to the respective global data stores. Once all data is stored in its respective stores, the design point has completely passed through the evaluation environment. The search algorithm can then acquire the evaluation results as desired.

## 4.2 Implementation

Transferring the design of the scalable evaluation workflow presented in the previous section to an actual working application involves a large amount of design choices and implementation details. This section will cover the major design decisions and infrastructure required to create a working scalable evaluation environment. The implementation of the scalable evaluation environment is based on Python (version 3.10). Python has been chosen for its quick prototyping, widespread support, and integration with a large variety of scientific packages.

### 4.2.1 Configuration

Each component in the workflow, design points, and the evaluation environment itself has many parameters and settings that can be configured, enabling a dynamically configurable workflow. In order to communicate this to the respective entity, JSON is utilised to generate structured configuration files. All workflow components and design points read and parse the corresponding files during their initialisation.

The configuration file for the workflow and its components is created before the environment is instantiated. Global configuration settings include the path where the search algorithm stores the simulation source files for each design point, the path for all three global output stores, and the relative paths to be used by all simulations (i.e., for local output). Currently, only the resource controller and design point queues require additional configuration settings. For the design point queues, a set containing identifiers, priorities, and scheduling policies is specified to represent the available queues. Configuration of the

## 4. METHODOLOGY

---

resource controller determines the platform that will be utilised and the settings of corresponding parameters, thereby facilitating a dynamically configurable workflow agnostic to its computing environment.

Each individual design point also has a configuration file detailing evaluation settings. Where the workflow configuration must be generated before the evaluation environment is instantiated, an individual design point configuration can be generated at runtime. In order to facilitate future usage of different simulation frameworks, addressing the simulator agnostic challenge, the configuration includes a field specifying the simulator to be utilised. Based on this field, the configuration file will contain further settings on the compilation and simulation of the design point, mostly concerning the usage of available command-line parameters. For instance, with OMNeT++ settings like PDES, a time limit, or external libraries can be passed through the configuration file. As configurations are highly dependant on the simulator and its capabilities, parsing the configuration file is based on the provided simulator parameter. By creating a simulator-specific parser, future integration of different simulators can be enabled by providing a corresponding parser. A simulator-specific parser reads the parameters present in the configuration file, captures only the parameters expected and valid for that simulator, and arranges the execution of the simulation corresponding to the parameter values.

### 4.2.2 Design Point Object

When the search algorithm creates a design point, it must instantiate a Design Point Object (DPO) defined by the evaluation environment. DPOs are used by the workflow components during communication, to manage the design points, and to execute the evaluations (i.e., when the manager sends work to the resource controller, it sends a list of DPOs). Each DPO represents a design point, of which its data is contained in a single folder, where the configuration file and all OMNeT++ simulation source files (excluding external libraries and external NED files) of the design point have to be present (i.e., the C++, ini, NED, and msg files). Creating a DPO makes communication between workflow components less complex, as often only the DPO has to be transferred.

Upon instantiation of a DPO, a Unique IDentifier (UID) is generated to manage and keep track of the DPOs present inside the evaluation environment. Currently, a basic UID scheme is utilised based on an MD5 hash of the folder contents from the respective DPO. In this basic scheme, the following is included in the hash (where subfolders are traversed recursively): (i) File names, (ii) Folder names (excluding the root folder of the DPO), and (iii) Content of files. However, effectively identical design points that maybe differ in

file structure or names will not be caught by the design point cache. Nevertheless, more advanced UID schemes can be adopted by the evaluation environment. Such a scheme, for example, could perform an in-depth analysis of the design point to improve the effectiveness of caching.

After the DPO has been initialised, the evaluation environment can start processing. Each DPO defines the compilation and simulation execution through its configuration file, which addresses the computing environment agnostic challenge. When a worker process calls the compilation or simulation method of a DPO, the corresponding configuration parameters are parsed. An exception is raised if a required parameter is absent or an invalid parameter value is given. Once the configuration has successfully been parsed and validated, the compilation or simulation will be performed.

To analyse the performance and behaviour of the evaluation environment, each DPO keeps track of runtime data, such as the start time of the simulation or the total time spent in the workflow. Additionally, an extensive logging system thoroughly records events during execution and all output generated by compilation and simulation is stored. By recording these diagnostics, the user can investigate and analyse problems that may arise during the evaluation of a DPO.

### 4.2.3 Distributed Computing

The main component that provides scalability to the evaluation environment is the resource controller. As shown in Figure 4.1, the resource controller has to manage the worker processes on the compute resources. The goal was to provide both simulation campaigns and PDES to the workflow without limiting or restricting the existing capabilities of a simulator. Simulation campaigns and PDES are approaches that address the workload-level and task-level parallelism challenges respectively. Development of the workflow was performed on a compute cluster using the Slurm Workload Manager, where users can reserve compute nodes from a head node (administrative file node).

During the implementation of the resource controller, many approaches were considered. An early version explored the use of MPI, where each workflow component and all workers are a separate process running on the compute resources. However, a significant development overhead was present to properly implement communication and synchronisation protocols, which limits the development of features of the evaluation environment. It was also recognised that the DSE would be limited by the Slurm reservation on the cluster, violating the aim to address the computing environment agnostic challenge. When the entire evaluation environment is on the compute resources, the state and data could be

## 4. METHODOLOGY

---

corrupted if the reservation is terminated before the execution is completed. Additionally, the environment could not be scaled adaptively during execution, as the number of compute resources is bounded by the reservation.

Instead of executing the entire environment on the compute resources, a different approach is employed where only the worker processes are on the compute resources. By moving the simulation environment away from the compute resources, the resource controller can dynamically adapt the number of compute resources through reservations on the compute cluster. It also mitigates the potential termination of the environment, as the workflow components are no longer bound by a reservation.

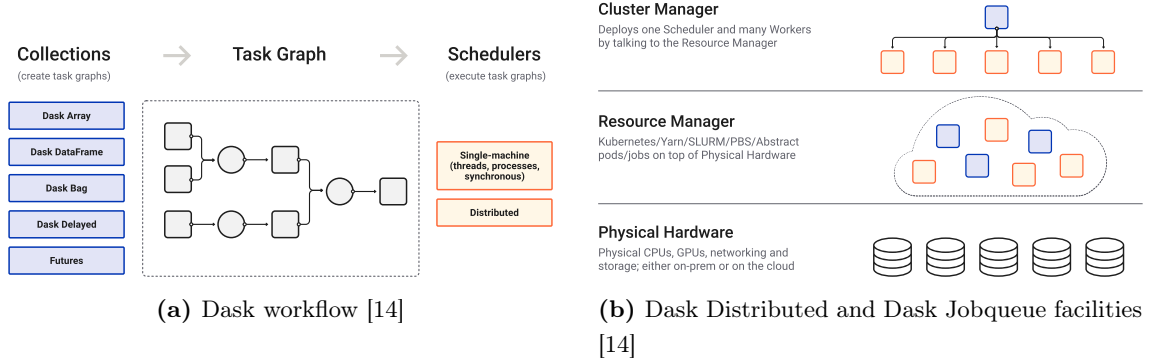
Another issue encountered in the early version using MPI was the development overhead of communication and synchronisation protocols limiting the development of evaluation environment features. Instead of building communication and synchronisation handlers from scratch, existing well-known task scheduling tools were investigated. Specifically, Celery, Luigi, and Dask were considered.

Celery is open-source software providing asynchronous task queues based on distributed message passing [12]. Although the focus is on real-time processing, task scheduling is supported. A workload is distributed across threads and machines using the task queue. The task queues are monitored by worker processes requesting work. A task is added to the task queue by clients. Communication in Celery is message-passing based, where a message broker is responsible for the transfer of messages between entities. Availability, redundancy, and horizontal scaling are supported by the Celery system, as multiple workers and brokers can be present. Additionally, Celery can be deployed on a range of systems, from a single machine to across compute clusters.

Luigi is an open-source project based on, hosted, and utilised by Spotify to orchestrate complex pipelines of batch jobs [39]. Luigi is also utilised by other well-known companies such as GIPHY, SeatGeek, Squarespace, and Red Hat. The intended purpose of Luigi is to address the orchestration and pipelining usually involved with long-running batch processes. During these processes, tasks have to be chained and automated, and failures are likely to occur. In order to focus on the tasks themselves and their dependencies, Luigi addresses workflow management and provides services such as dependency resolution, visualisation, handling failures, and command-line integration.

Dask is an open-source Python library for parallel computing and is supported by and utilised by many companies, such as Anaconda, Microsoft, NASA, NVIDIA, and Shell [14] [15]. Similarly to Celery and Luigi, it provides dynamic task scheduling optimised for computation. Dask has multiple plugins for large-scale distributed computing, such as Dask

## 4.2 Implementation

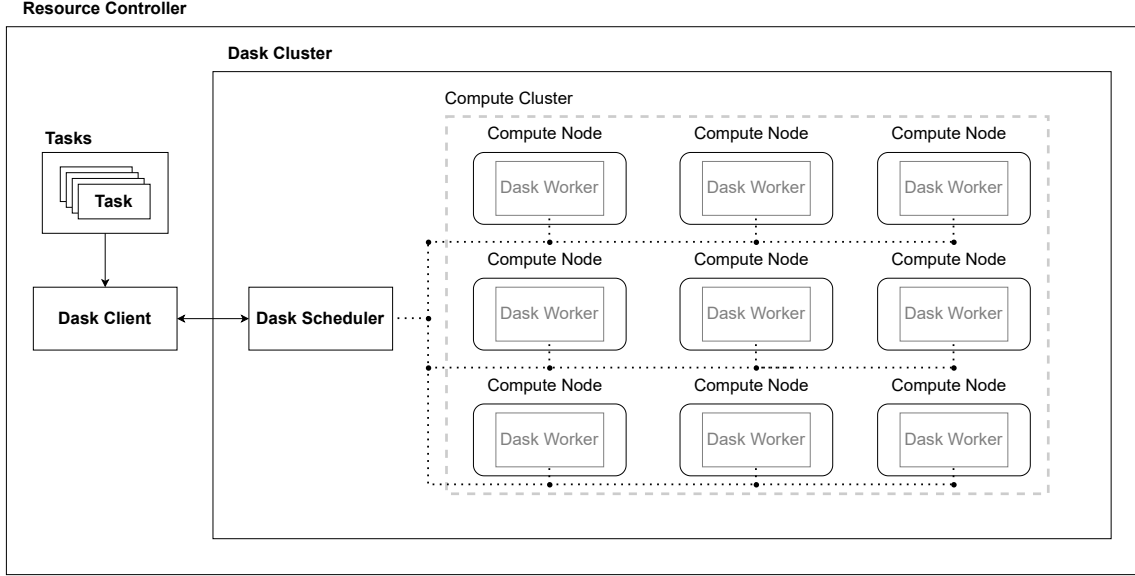


**Figure 4.2:** Dask, Dask Distributed, and Dask Jobqueue overview.

Distributed [17] and Dask Jobqueue [16]. Dask Distributed provides a centrally managed, distributed, dynamic task scheduler to extend Dask APIs to moderate-sized clusters. The scheduler is an asynchronous event-driven process managing a collection of Dask worker processes spread across multiple machines. Tasks submitted to the scheduler are Python functions operating on Python objects. The scheduler keeps track of task execution and scheduling using a directed acyclic graph of tasks, where dependencies and priorities are integrated. Dask Jobqueue enables users to deploy Dask on job queuing systems typically found in high-performance supercomputers, academic research institutions, and other clusters. By combining Dask Distributed and Dask Jobqueue, Dask can provide an interface in Python for dynamic task scheduling on compute clusters.

After careful consideration of and experimentation with the previously discussed scheduling tools, Dask was chosen for its user-friendly interface in Python, widespread support, active development, integration with CMS through Dask Distributed and Dask Jobqueue, and the ability to adaptively scale compute resources. By integrating Dask into the resource controller, the workflow further facilitates the aims of dynamic configurability and compute environment agnosticism. Figure 4.2 visualises the workflow of Dask and the capabilities of Dask Distributed with Dask Jobqueue. Dask has its own built-in collections, mimicking well-known structures such as the array and dataframe, which are combined into a task graph to be scheduled on the available resources. Within Dask, arbitrary task scheduling is performed using Future objects. A Future represents the execution of the arbitrary task and contains runtime data such as status and result. As the Dask workflow supports both local-machine scheduling and cluster-level scheduling through CMS, it provides a common interface to transfer between prototyping on a local machine and a compute cluster for deployment.

## 4. METHODOLOGY

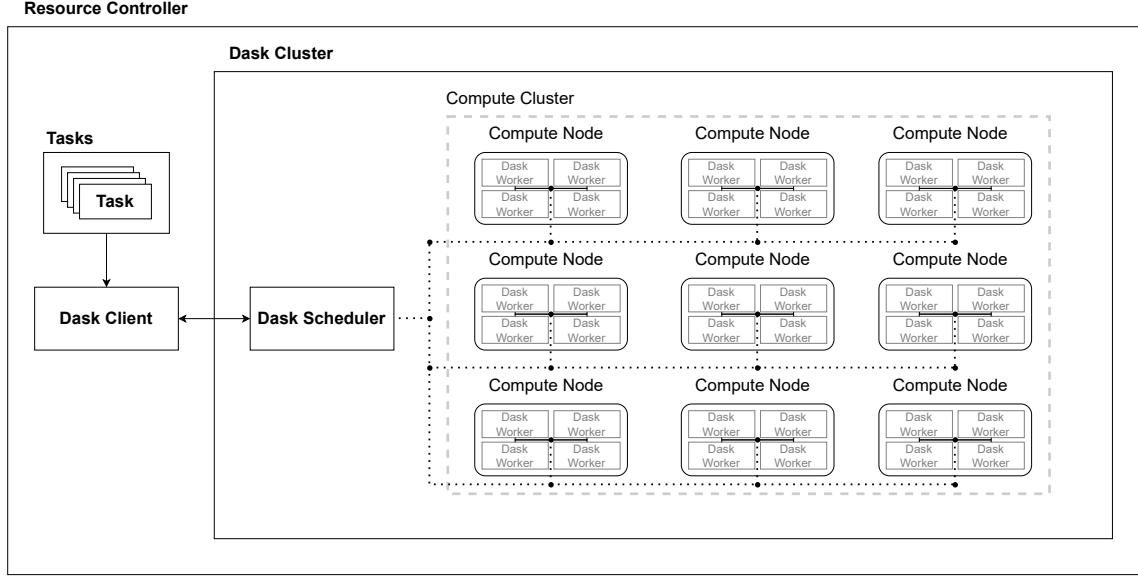


**Figure 4.3:** Resource Controller internal workflow with a single Dask Worker per Compute Node.

Dask is integrated into the resource controller by creating a Dask Distributed Client and a Dask Cluster for the CMS using Dask Jobqueue. A Dask Cluster object can allocate resources on the cluster using the CMS and manage worker processes on the resources using an internal scheduler. The Dask Client is the interface through which tasks are accepted and forwarded to the worker processes through the internal Dask Scheduler of the Dask Cluster. A worker process in Dask can manage multiple tasks at the same time. For example, all simulations executing on a compute node can be managed by a single worker, or each simulation could be managed by its own worker. Different configurations could be optimal depending on the workload characteristics and the hardware platform. The internal workflow of the resource controller using Dask Distributed with Dask Jobqueue is demonstrated in Figures 4.3 and 4.4. Upon initialisation of the Dask Cluster, the required hardware specification for a single job is specified, which is used by the Dask Scheduler to request resources through the CMS. As the Dask Cluster uses the job specification to scale to the number of desired resources, a single job should not surpass the specifications of a single machine in the compute cluster.

In the evaluation environment, a task is defined as a Python function that initiates the compilation and simulation of a DPO. By defining the evaluation process inside a DPO, the aims of simulator agnosticism and task-level parallelism are facilitated. When presenting a task to the Dask Client, a corresponding DPO is provided. By default, a



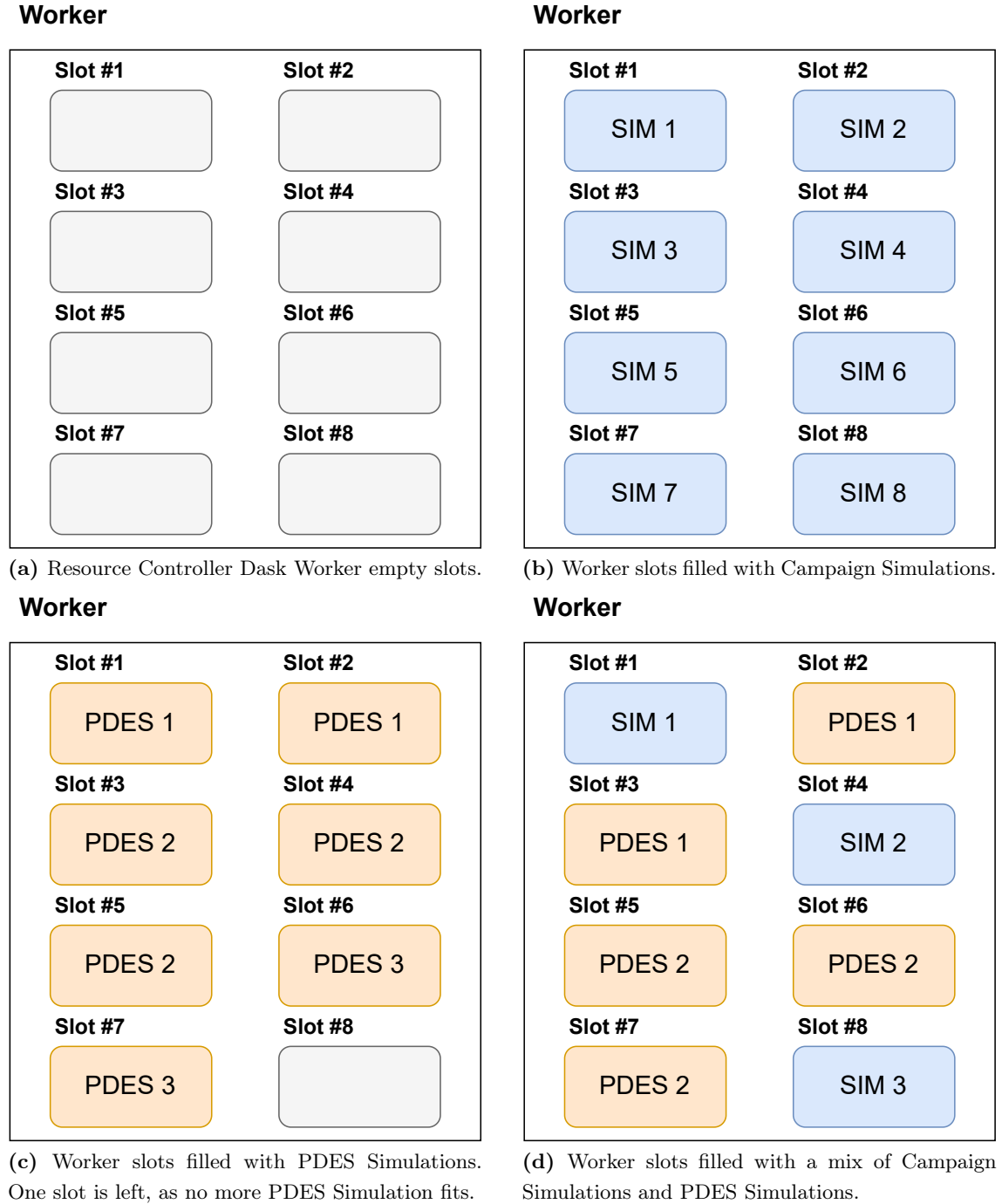


**Figure 4.4:** Resource Controller internal workflow with four Dask Workers per Compute Node.

single task will run on a single core of a compute node, which fits with the Simulation Campaign methodology. However, the simulation environment also needs to be capable of providing task-level parallelism (i.e., PDES), which can utilise more than a single core at a time. In Dask, task-level parallelism can be enabled through the resources tag. During the initialisation of a Dask Cluster, the availability of scarce resources (like GPUs) can be indicated for a worker. Whenever a task requires such a resource, it can be listed as a requirement in the submission through the Dask Client. A task can only be scheduled to a worker when the resource requirement can be satisfied. The resource annotation can be utilised to enable task-level parallelism, like PDES, by indicating the number of slots available to a worker. Slots can generally be correlated to the number of cores available in a processor (i.e., a single worker on a 32-core processor would have 32 slots as resources available, or more if simultaneous multithreading is available). Upon submission of a DPO requiring PDES, the number of logical processes in the PDES instance will be set as the required number of slots. When a worker has the required number of slots available, the task is scheduled and the worker will update its number of available resources. Figure 4.5 visualises the use of slots in a worker to enable the use of PDES. This strategy also enables the use of the built-in parameter study of OMNeT++ and other methodologies requiring task-level parallelism or specialised hardware. Furthermore, using the resources annotation allows the evaluation environment to simultaneously support the Simulation Campaign and

## 4. METHODOLOGY

task-level parallelism methodologies. However, a PDES simulation is limited to utilising at most all resources on a single compute node, as a Dask Worker process does not span multiple compute nodes. Additionally, as shown in Figure 4.5c, resources are left unutilised when a PDES task no longer fits in the currently available slots.



**Figure 4.5:** Resource Controller Dask Worker slots.

## 5

# Evaluation

After having proposed a scalable workflow for an evaluation environment to be utilised in DSE for dCPS, the evaluation section will investigate its behaviour, performance, and applicability. In order to evaluate the scalability of the environment, a (computing) platform and system models are introduced in the setup. After the setup, the experiment definitions are presented. An experiment definition covers various aspects by detailing the goal, corresponding research questions, hypotheses, metric collection, specialised setup factors, and the corresponding analysis. To finish the evaluation chapter, the results of the experiments are presented.

## 5.1 Setup

Before the experiments can be defined and performed, the setup for the evaluation needs to be discussed. The evaluation setup will discuss the specifications of the chosen computing platform, the versions of the critical software components, and the system models used throughout the evaluation.

### 5.1.1 Platform

A distributed computing platform is imperative for the evaluation of a scalable environment. The DAS-6 compute cluster was utilised during the evaluation to perform the experiments. Specifically, the UvA-SNE cluster site was host to the operation. As briefly alluded to in Chapter 2, DAS-6 is the sixth iteration of Distributed ASCI Supercomputer (DAS) and part of the cluster computing initiative by the Advanced School for Computing and Imaging (ASCI). ASCI involves Dutch universities and research institutes [5]. Mul-

## 5. EVALUATION

---

multiple cluster sites are located at universities and research institutes, allowing researchers and students to perform research on compute clusters.

DAS-6 contains both single- and dual-socket compute nodes, built by Lenovo [13] [5]. The most common CPU for the single-CPU compute node configuration is the 24-core AMD EPYC-2 (Rome) 7402P CPU, and for the dual-CPU compute node configuration there are two 16-core AMD EPYC-2 (Rome) 7282 CPUs. DAS-6 itself is a distributed system spread over six clusters, which are located at five physical sites. The configuration of DAS-6 differs per cluster. In the case of the UvA-SNE cluster site, it contains eight dual 16-core compute nodes with a base-clock of 2.8 GHz and each 128 GB of memory. Between nodes, there is a 100 Gbit/s RoCE Ethernet interconnect. Four of the compute nodes are equipped with a single NVIDIA A4000 or A5000 GPU. However, the GPUs themselves will not be utilised throughout the evaluation process. The other four compute nodes contain a large number of disks intended to be used for IO applications. During the evaluation process, up to four of the non-IO-specialised compute nodes will be utilised to prevent occupying the entire UvA-SNE cluster and its specialised nodes. The CMS utilised by the DAS-6 system is Slurm [63].

Part of the evaluation platform is the software components involved. The scalable environment utilised various software packages to provide its service. As stated in Chapter 4, Python 3.10 is the programming language of choice. As such, well-known modules from the Python standard library, like *os*, *time*, *logger*, *subprocess*, and *json* are used. A considerable part of the scalability of the environment is provided by the resource controller. The resource controller makes use of Dask, Dask Distributed, and Dask Jobqueue to distribute the evaluation of design points on an HPC cluster.

### 5.1.2 Models

In order to perform experiments on the scalable evaluation environment, a set of system models is required. The system models will be utilised to represent design points within a DSE process. Firstly, related work on dCPS or CPS system models in OMNeT++ was sought after, as this could provide the desired simulation behaviour in the context of DSE2.0. However, such a system model suitable for the evaluation process was not found. Instead, a large Ethernet simulation model from the SPEC CPU® 2017 benchmark suite [1] (specifically, the model is part of its Integer suite) was chosen as a baseline. The model is included in INET, an open-source OMNeT++ model suite for wired, wireless and mobile networks. Specifically, it is the LANs model from the Ethernet examples set. The model represents a large Ethernet campus backbone, see Figure C.1, containing around

8000 computers and 900 switches and hubs. In the model, various Ethernet technologies are mixed (100 Mb full-duplex, 10 Mb UTP, switched hubs, and more). Multiple levels of medium and smaller networks are interconnected and attached to the backbone. The model is configured to simulate 120 seconds of simulated time. Henceforth, this system model will be referred to as *INET-LANS*.

While INET-LANS can be used throughout simulation campaigns, it is not applicable to the PDES methodology. The INET suite was not built with parallelisation in mind and certain modules violate the PDES constraints [59] in OMNeT++ [56]. A different system model is required to perform evaluations of the scalable environment in this aspect. The capabilities of the PDES methodology will be evaluated using three different models, each representing one of the following system behavioural traits: (i) Compute Intensive, (ii) Communication Intensive, and (iii) Subsystem Intensive. Specifically, compute intensive refers to a system which can be split into distinct logical processes, where the logical processes are dominated by computation and relatively little communication occurs between the logical processes. On the contrary, communication intensive refers to a system which can be split into distinct logical processes, where the logical processes are dominated by communication and relatively little computation occurs within the logical processes. A composite of the two is subsystem intensive, where the logical processes contain multiple systems, which can have a balance of communication and computation, and between the logical processes there is communication (less prevalent as with communication intensive, but present nevertheless).

Utilising distinct system models would complicate the evaluation of results between the different behavioural PDES traits, as the system itself may significantly influence the performance and behaviour. In order to prevent such complications, a single system that can be configured to demonstrate the three different traits is preferred. One such system is the Closed Queueing Network (CQN) model, which is also utilised by OMNeT++ to demonstrate the concept of PDES [59] [45] (OMNeT++ version 6.0.1, CQN sample). Figure 5.1a illustrates the CQN system model. Inside the CQN, there are modules of Chained Queues (CQs). Each CQ consists of a Switch (illustrated by the blue circles) and a sequence of queues (illustrated by the yellow boxes). Whenever a job arrives in a switch, it forwards the job, at random, to one of the CQs, where it arrives at the first queue in the CQ its sequence of queues. Built into the switch is a retention rate, which indicates the percentage of jobs that should be retained in a CQ (as the retention of a job is based on random samples, it may not match the retention rate exactly). The retention rate can be used to increase the volume of communication between logical processes. Communication

## 5. EVALUATION

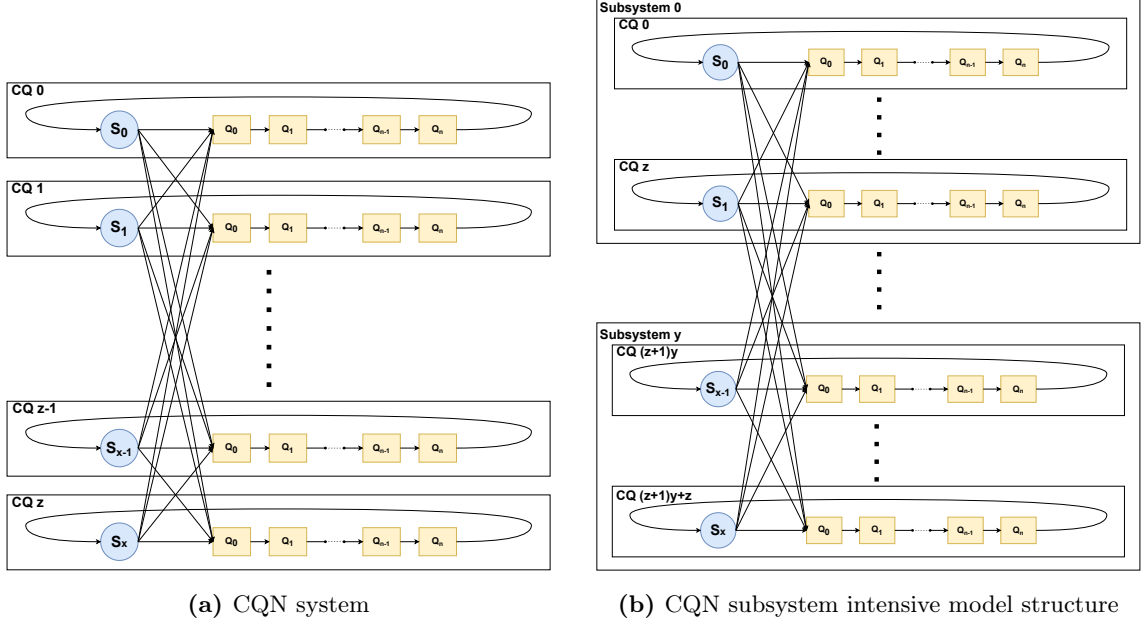
---

channels between a switch and all other queues are illustrated using a one-way arrow in Figure 5.1a. When a job arrives in a queue, computation is simulated as a random wait (based on an exponential distribution). Once a queue has processed a job, it forwards the job to the next queue in sequence. The last queue in the sequence forwards the job to the corresponding switch of the current CQ. Communication channel delays can be set globally for the channels between a switch and a queue, and between two queues (where the connection between the last queue and the switch is the same as between two queues). Additionally, a predefined number of initial jobs can be set to be present in the queues. Henceforth, this system model will be referred to as *CQN*.

As is illustrated in Figure 5.1a, the CQN model can be scaled in two ways, where a single CQ can scale up or down the number of queues, and the CQN can scale up or down the number of CQs. Combining the flexibility in the model structure with the configurability of processing and communication channel delays enables the modelling of the three different PDES traits using the CQN system. During a PDES simulation, the system model is divided into logical processes by the CQs (i.e., each CQ will be its own independent logical process).

The compute, communication, and subsystem intensive systems will be modelled as described in Table 5.1. As can be seen, both the compute and communication intensive models will consist of eight CQs, each containing a thousand queues long sequence. The compute intensive system has a retention rate of 0.95, indicating that only 5% of the jobs will be forwarded to a different CQ. On the contrary, the communication intensive system has a retention rate of 0.1, where 90% of the jobs are forwarded to a different CQ. Thereby, the communication intensive model will generate, relative to the computation intensive model, a lot of communication (and therewith, synchronisation) between logical processes. Additionally, the communication delay between a switch and a queue is set to ten seconds for the computation intensive system and 0.1 seconds for the communication intensive system, creating a low-frequency, high-latency communication channel and high-frequency, low-latency communication channel, respectively, between logical processes. For both the computation and communication intensive models, the queue-queue channel delay and the queue processing time are 0.1 seconds. It should be noted that the delay between queues, between a switch and a queue, and the processing time are reported as the average of an exponential distribution. In the computation intensive system, the number of initial jobs is set to 10, and for the communication intensive system, the number of initial jobs is set to 1000. By increasing the number of jobs, the amount of communication between logical processes is further increased, relative to the computation intensive system, for the

## 5.2 Experiment Definition



**Figure 5.1:** CQN model structure.

communication intensive model. Concerning the subsystem intensive model, the setup slightly deviates in structure from the previous two models. Figure 5.1b illustrates the subsystem intensive model, where the logical processes will contain multiple CQs. Each of the eight logical processes will contain 10 CQs, resulting in 80 CQs in total for the CQN. Inside each CQ, 100 queues will be present with a retention rate of 0.9. For the delays between a switch and a queue, between a queue and a queue, and the processing delay inside a queue, the same parameter values as for the computation intensive system holds (i.e., 10 seconds, 0.1 seconds, and 0.1 seconds respectively). The number of initial jobs is set to 100 for each queue.

## 5.2 Experiment Definition

After having described the evaluation setup, the experiment definitions can be detailed. An experiment definition characterises the various aspects, such as research questions, goals, metrics, and any additional setup, involved in the evaluation. This section is divided into the following three experiment definitions: (i) Simulation Campaign, (ii) Worker Size, and (iii) PDES.

## 5. EVALUATION

**Table 5.1:** Configuration parameters of the CQN models.

	Comp. Intensive	Comm. Intensive	Sub. Intensive
<b>CQs</b>	8	8	80
<b>Queues</b>	1000	1000	100
<b>Retention Rate</b>	0.95	0.1	0.9
<b>S-Q Delay (s)</b>	10	0.1	10
<b>Q-Q Delay (s)</b>	0.1	0.1	0.1
<b>Process Time (s)</b>	0.1	0.1	0.1
<b>Initial Jobs</b>	10	1000	100
<b>Subsystem Size</b>	N/A	N/A	10

### 5.2.1 Experiment: Simulation Campaign

The evaluation process will start with the simulation campaign experiment, which focuses on the behaviour of the environment and the performance of the simulation campaign methodology. Comparisons against various baselines, such as sequential order simulation, will be made to analyse the effectiveness of the solution. By investigating the performance and potential overhead in the workflow, an assessment can be made of the utility that can be provided by the environment to DSE for dCPS.

The simulation campaign experiment will support the exploration of Research Questions 1.1, 1, and 1 by analysing the behaviour and performance of applying the simulation campaign methodology. Investigating the campaign methodology can show whether simulation campaigns are applicable, viable, performant, and advisable to be utilised in DSE for dCPS. Additionally, the evaluation of the simulation campaign could uncover potential adaptations or other considerations that need to be accounted for in future work when implementing a distributed evaluation workflow to address scalability in DSE for dCPS.

In order to answer the corresponding research questions and create a baseline view of the performance and utility provided by the environment, multiple sets of evaluations are performed using the INET-LANS model. As this model provides a computationally demanding simulation, representing the demanding task of evaluating a complex dCPS design point in a DSE process, it will grant insight into the applicability of the proposed solution. During the evaluation, various metrics will be captured to address the previously stated goals. Execution time is one of the metrics paramount to the analysis of the proposed solution. However, the execution time can be considered at specific timeframes (i.e., spe-



## 5.2 Experiment Definition

---

cific starting and end points). During the simulation campaign experiment, the following execution time timeframes will be captured and utilised during the analysis: (i) *Environment Time*, which is registered from the start of the initialisation up to the termination of the environment. This time frame includes the initialisation of the environment, creation of the DASK cluster, resource reservations through the CMS, evaluations in the environment, clean up of the environment, and the termination of the environment. Henceforth, equations will refer to the *Environment Time* as  $T_{\text{campaign}}$ . (ii) *Simulation Time*, which is registered from the start up to the end of the simulation task performed by a DPO. This time frame includes parsing the DPO configuration, execution of the simulation, and storing the corresponding data and logs. (iii) *Compilation Time*, which is registered from the start up to the end of the compilation task performed by a DPO. This time frame includes parsing the DPO configuration, creation and execution of a Makefile, and storing the corresponding data and logs. (iv) *Worker Time*, which is registered from the start up to the end of a single task in the evaluation environment. This time frame concerns the execution of an evaluation by a Dask Worker and thereby includes, amongst others, the Simulation Time and Compilation Time of the corresponding DPO. For all timeframes listed, the environment records the total time required and the timestamps of the start and end points.

The other metrics present in the simulation campaign experiment are speedup and efficiency, which are metrics derived from the execution time. Speedup  $S$  is defined as the ratio of execution time improvement for a solution  $T_s$  compared to a specific baseline  $T_b$ , as shown in Equation (5.1). In order to assess the efficiency, the speedup is compared against an ideal speedup. The ideal speedup is based on the increase in computing resources (i.e., going from a single-core setup to 32 cores yields an ideal speedup of 32). The efficiency and speedup can be calculated using a variety of baselines. During the simulation campaign experiment, the following baselines will be utilised: (i) *Sequential*, which represents the (average) execution time required to perform a single sequential evaluation of the system model. The execution time is measured as the time it takes to perform the compilation of the model and the simulation through OMNeT++. An average execution time is captured by, in sequence, performing ten sequential system evaluations. The evaluation environment is not involved in this baseline. Henceforth, equations will refer to the *Sequential* baseline time as  $T_{\text{seq}}$ . (ii) *Sequential Multi*, which represents, like *Sequential*, the (average) execution time required to perform a single sequential evaluation of the system model. The execution time is measured as the time it takes to perform the compilation of the model and the simulation through OMNeT++. However, where *Sequential* captures the average

## 5. EVALUATION

---

execution time by running evaluations in sequence, *Sequential Multi* performs multiple (number is equal to the number of available cores) system evaluations concurrently on the same computing resource. By performing multiple evaluations concurrently, the simulation campaign behaviour is mimicked and other runtime artefacts (i.e., context switching and OS interruptions) will be captured. The evaluation environment is not involved in this baseline. Henceforth, equations will refer to the *Sequential Multi* baseline time as  $T_{\text{seq\_multi}}$ .

By measuring the *Environment Time* ( $T_{\text{campaign}}$ ) and the baseline timeframes, Equations (5.2)-(5.3) can be utilised to calculate a speedup and efficiency of the solution for various runtime configurations, where  $n$  indicates the number of models evaluated in the simulation campaign, and  $r$  indicates the number of resources available (i.e., 32 for a single DAS-6 compute node, as it contains 32 CPUs). Both the speedup and efficiency can provide insights into the applicability and potential overhead when employing the solution in different runtime configurations.

$$S = \frac{T_b}{T_s} \qquad E = \frac{S}{r} \qquad (5.1)$$

$$S_{\text{seq}} = \frac{T_{\text{seq}} * n}{T_{\text{campaign}}} \qquad E_{\text{seq}} = \frac{S_{\text{seq}}}{r} \qquad (5.2)$$

$$S_{\text{seq\_multi}} = \frac{T_{\text{seq\_multi}} * n}{T_{\text{campaign}}} \qquad E_{\text{seq\_multi}} = \frac{S_{\text{seq\_multi}}}{r} \qquad (5.3)$$

The simulation campaign experiments will be performed from 1 up to 4 nodes (limit imposed by the UvA-SNE DAS-6 cluster site), where a single worker process per nodes is present. Both weak and strong scaling (as discussed in Section 2.3) will be investigated, to investigate how the number of compute resources affects the speedup and efficiency of the solution. Additionally, the performance behaviour when increasing the number of evaluations for a fixed number of compute resources is analysed.

Throughout the simulation campaign experiment, the maximum number of evaluations performed concurrently on a single compute node is equal to the number of cores available in a single compute resource. However, the compute resources in the UvA-SNE cluster have simultaneous multithreading (SMT) available, allowing two threads to be executed on a single CPU core. By utilising SMT, the proposed solution can be scaled to 64 evaluation on a single compute node. Nevertheless, sharing a single core could also potentially lead to degradation of performance. The same runtime configurations described before will

also be executed using SMT in order to investigate the applicability during a simulation campaign. Besides comparing execution times, Equation (5.4) demonstrates a relative speedup  $S_{rel}(x, y)$ , where the SMT and non-SMT models are compared to analyse the relative performance gain or degradation. In Equation (5.10),  $x$  and  $y$  indicate the use of the SMT enabled or non-SMT version used in the comparisons, and  $S_x$  and  $S_y$  indicate the corresponding speedup of the version, which can be based on the *Sequential* and *Sequential Multi* execution time baselines.

$$S_{rel}(x, y) = \frac{S_x}{S_y} - 1 \quad (5.4)$$

### 5.2.2 Experiment: Worker Size

The evaluation process continues with the worker size experiment, which focuses on the distribution of evaluations by the resource controller and the corresponding performance impact of the simulation campaign methodology. Comparisons against two sequential baselines will be made to analyse the potential performance improvement or degradation when scaling the number of workers utilised in the solution. By investigating the potential overhead in the resource controller, an assessment can be made of the resource distribution configuration for the environment in the context of DSE for dCPS.

The worker size experiment will support the exploration of Research Questions 1.1, 1, and 1 by analysing the behaviour and performance impact of scaling the number of workers on a single compute node. By investigating the impact of scaling the worker size, insights can be gained into the key design choices in the distributed workflow management and their impact on the scalability and efficiency of the solution. Additionally, the evaluation of the worker size scalability could uncover potential adaptations or other considerations that need to be accounted for when employing the solution for different use cases or on different HPC systems, where it needs to dynamically adapt to address the changing requirements and needs for different DSE environments.

During the simulation campaign experiment, the number of workers per compute resource was fixed to one. However, the number of workers can be scaled, potentially alleviating the administrative workload, which could lead to an improvement in evaluation latency and overall throughput. Nevertheless, increasing the number of workers per compute resource could also have a negative impact on performance, as more processes will be running on a single compute node, and more communication and synchronisation is present within the resource controller.

## 5. EVALUATION

---

Similarly to the simulation campaign experiment, the INET-LANS model will be utilised to represent an evaluation in the DSE of dCPS. Additionally, the *Simulation Time*, *Compilation Time*, *Worker Time*, and *Environment Time* are recorded to investigate the impact of different resource controller configurations. In order to analyse the behaviour, comparisons will be made against the *Sequential* and *Sequential Multi* baselines, as described in the definition of the simulation campaign experiment.

The worker size experiments will be performed from 1 up to 4 nodes (limit imposed by the UvA-SNE DAS-6 cluster site), where the number of worker processes per compute node is scaled by powers of two (i.e., 1, 2, 4, 8, 16, 32). Similarly to the simulation campaign experiment, both weak and strong scaling (pertaining to the number of compute nodes) will be covered. The workload per worker process is determined by the weak and strong scaling primitive of the compute nodes, where the workload, given a fixed number of workers, stays the same with weak scaling, but decreases with strong scaling. Furthermore, when the number of compute nodes is fixed, the workload per worker process increases when the number of workers decreases, and the workload per worker process decreases when the number of workers increases.

### 5.2.3 Experiment: PDES

Both the simulation campaign and worker size experiments concerned the simulation campaign methodology of the proposed evaluation environment. During this experiment, the focus is shifted towards the PDES methodology, which is also available in the proposed solution. As such, the Compute, Communication, and Subsystem Intensive configurations of the CQN model, detailed in Table 5.1, will be utilised throughout the evaluation process. By investigating the performance impact of different behavioural traits in system models, insights can be gained on the applicability of the PDES methodology for different kinds of systems. The DSE process might prefer the PDES methodology over the simulation campaign methodology, depending on the system model itself.

The PDES experiment will support the exploration of Research Questions 1.1, 1, and 1 by analysing the behaviour and performance of applying the PDES campaign methodology, as opposed to utilising the sequential simulation campaign methodology. The evaluation of the PDES methodology investigates the applicability, viability, performance, and advisability of utilising it in DSE for dCPS. The evaluation of PDES will also provide insights into the use of PDES on a variety of system model behavioural classes, which can guide a designer in deciding whether PDES is advisable for all models in their DSE for dCPS (or only for a subset of the models). Additionally, the evaluation of the PDES could uncover potential

## 5.2 Experiment Definition

---

adaptations or other considerations that need to be accounted for when implementing a distributed evaluation workflow to address scalability in DSE for dCPS.

Similarly to the simulation campaign and worker size experiments, the *Simulation Time*, *Compilation Time*, *Worker Time*, and *Environment Time* are the main execution time timeframes recorded. Based on these timeframes, the speedup and efficiency metrics are calculated. During the PDES experiment, a variety of baselines will be utilised to calculate the efficiency and speedup. Included in the baselines are the  $T_{\text{seq}}$  and  $T_{\text{seq\_multi}}$  already discussed in Subsection 5.2.1. The following additional baselines will be present in the PDES experiment: (i) *Sequential Campaign*, which represents the (average) execution time required to perform a single campaign that performs an equal amount of sequential system model evaluations. The execution time is measured as the time it takes to perform the entire campaign, from the initialisation up to the shutdown of the evaluation environment. An average execution time is captured by performing five individual simulation campaigns, each utilising an independent evaluation environment. Henceforth, equations will refer to the *Sequential Campaign* baseline time as  $T_{\text{seq\_campaign}}$ . (ii) *Sequential PDES*, which represents the (average) execution time required to perform a single PDES evaluation of the system model. The execution time is measured as the time it takes to perform the compilation of the model and the simulation through OMNeT++. An average execution time is captured by, in sequence, performing 10 PDES system evaluations. The evaluation environment is not involved in this baseline. Henceforth, equations will refer to the *Sequential PDES* baseline time as  $T_{\text{seq\_pdes}}$ . (iii) *Sequential PDES Multi*, which represents the (average) execution time required to perform a single PDES evaluation of the system model. The execution time is measured as the time it takes to perform the compilation of the model and the simulation through OMNeT++. However, where *Sequential PDES* captures the average execution time by running PDES evaluations in sequence, *Sequential PDES Multi* performs multiple (the number is equal to the number of PDES models that fit on the available cores) PDES system evaluations concurrently on the same computing resource (which is repeated ten times). By performing multiple evaluations concurrently, the simulation campaign behaviour is mimicked and other runtime artefacts (i.e., context switching and OS interruptions) will be captured. The evaluation environment is not involved in this baseline. Henceforth, equations will refer to the *Sequential PDES Multi* baseline time as  $T_{\text{seq\_pdes\_multi}}$ .

By measuring the *Environment Time* ( $T_{\text{campaign}}$ ) and the baseline timeframes, Equations (5.5)-(5.9) can be utilised to calculate a speedup and efficiency of the solution for various runtime configurations, where  $n$  indicates the number of models evaluated in the

## 5. EVALUATION

---

simulation campaign,  $r$  indicates the number of resources available (i.e., 32 for a single DAS-6 compute node, as it contains 32 CPUs), and  $lp$  indicates the number of LPs in the PDES model. Both the speedup and efficiency can provide insights into the applicability and potential overhead when employing the solution in different runtime configurations.

$$S_{\text{seq}} = \frac{T_{\text{seq}} * n}{T_{\text{campaign}}} \quad E_{\text{seq}} = \frac{S_{\text{seq}}}{r} \quad (5.5)$$

$$S_{\text{seq\_multi}} = \frac{T_{\text{seq\_multi}} * n}{T_{\text{campaign}}} \quad E_{\text{seq\_multi}} = \frac{S_{\text{seq\_multi}}}{r} \quad (5.6)$$

$$S_{\text{seq\_campaign}} = \frac{T_{\text{seq\_campaign}}}{T_{\text{campaign}}} \quad E_{\text{seq\_campaign}} = \frac{S_{\text{seq\_campaign}}}{1} \quad (5.7)$$

$$S_{\text{seq\_pdes}} = \frac{T_{\text{seq\_pdes}} * n}{T_{\text{campaign}}} \quad E_{\text{seq\_pdes}} = \frac{S_{\text{seq\_pdes}}}{\frac{r}{lp}} \quad (5.8)$$

$$S_{\text{seq\_pdes\_multi}} = \frac{T_{\text{seq\_pdes\_multi}} * n}{T_{\text{campaign}}} \quad E_{\text{seq\_pdes\_multi}} = \frac{S_{\text{seq\_pdes\_multi}}}{\frac{r}{lp}} \quad (5.9)$$

Besides comparing execution times, speedup, and efficiency, Equation (5.10) demonstrates a relative speedup  $S_{\text{rel}}(x, y)$ , where two system models are compared to analyse the relative performance gain or degradation. In Equation (5.10),  $x$  and  $y$  indicate the two system models used in the comparisons, and  $S_x$  and  $S_y$  indicate the corresponding speedup of the system models, which can be based on the *Sequential*, *Sequential Multi*, *Sequential Campaign*, *Sequential PDES*, and *Sequential PDES Multi* execution time baselines.

$$S_{\text{rel}}(x, y) = \frac{S_x}{S_y} - 1 \quad (5.10)$$

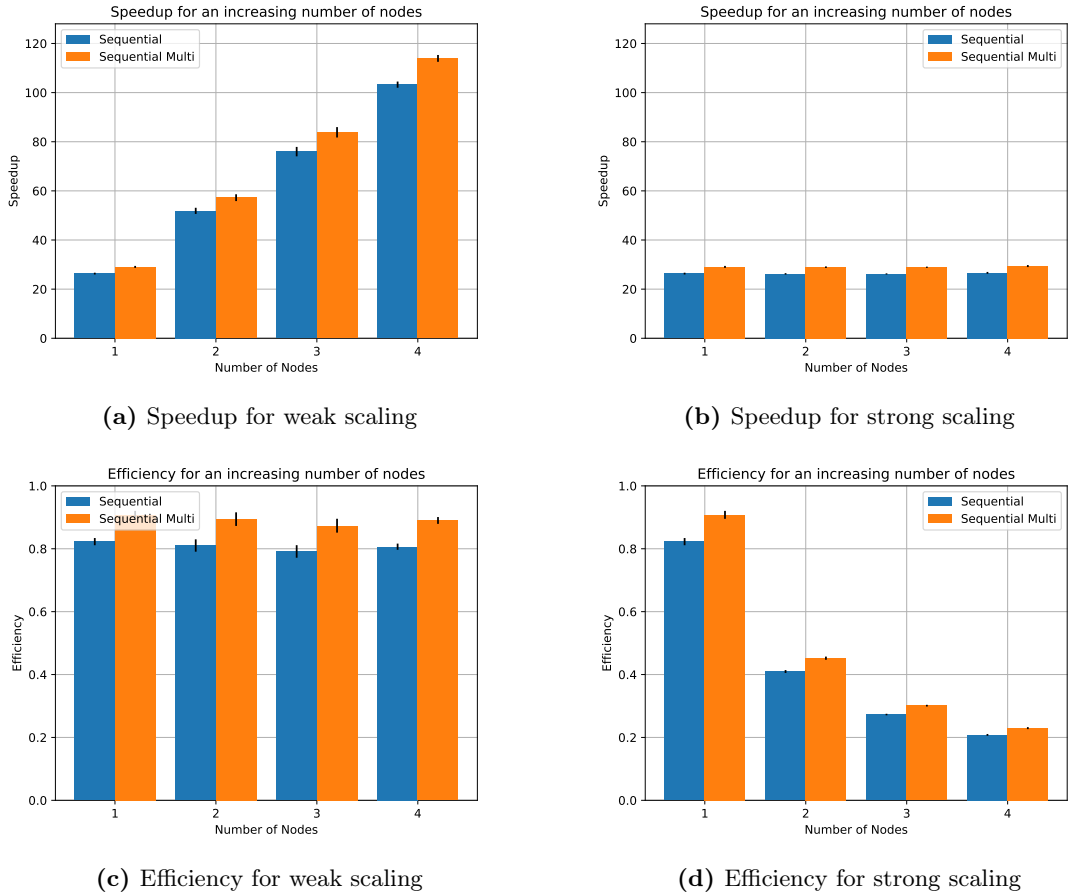
The PDES experiments will be performed from 1 up to 4 nodes (limit imposed by the UvA-SNE DAS-6 cluster site), where a single worker process per node is present. Both weak and strong scaling (as discussed in Section 2.3) will be investigated, as was the case for the simulation campaign experiment, to analyse how the number of compute resources affects the speedup and efficiency of the solution. By comparing the speedup and efficiency for the three different CQN system models, the behaviour, performance, and applicability of the PDES methodology for DSE of dCPS can be analysed.

### 5.3 Results

Based on the experiment definitions presented in the previous section, evaluations were performed according to their respective specifications. All unprocessed data collected and

utilised, together with further visualisations, is presented in Appendix A and B. Analysis and interpretation of the data will be part of Chapter 6. Nevertheless, this section will present the experiment outcomes and highlight aspects of the data.

### 5.3.1 Simulation Campaign

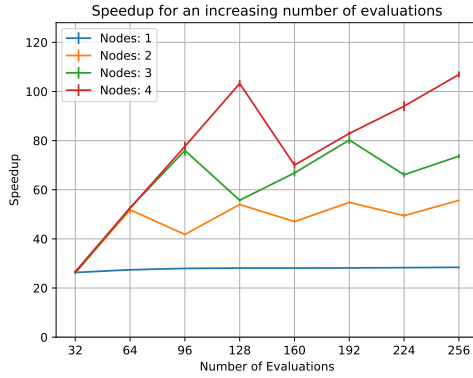


**Figure 5.2:** Efficiency and Speedup for strong and weak scaling with 32 evaluations as baseline for the single node configuration and the *Sequential* and *Sequential* execution time baselines.

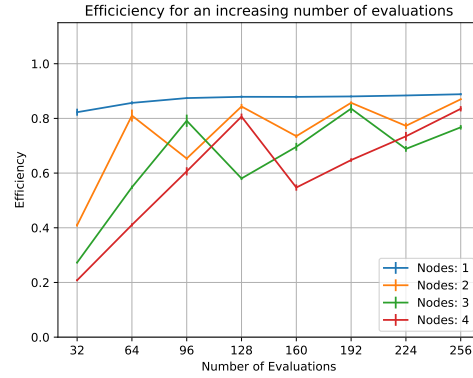
The first experiment performed in the evaluation chapter is the simulation campaign experiment, with the goal of analysing the effectiveness of the solution and utility that can be provided by the environment to DSE for dCPS. Firstly, data corresponding to the speedup and efficiency in different node configurations for both weak and strong scaling is displayed in Figure 5.2. The baseline number of evaluations for a single node is 32, where weak scaling scales the number of evaluations accordingly (i.e., 64 for two nodes, 96 for three nodes,

## 5. EVALUATION

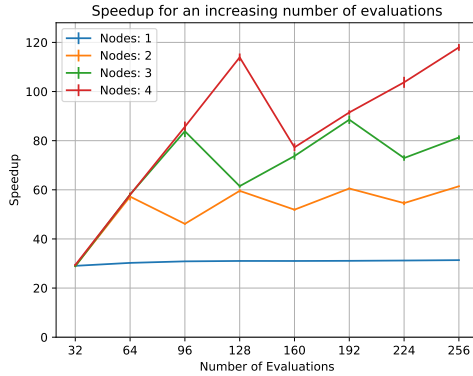
and 128 for four nodes), and strong scaling keeps the number of evaluations consistent for all node configurations. Two bars per node configuration are shown for the speedup and efficiency bar plots, where each bar represents the speedup and efficiency for a different baseline (in this case, the *Sequential* and *Sequential Multi* execution time baselines). Weak scaling demonstrates a steady increase in speedup and consistent efficiency, whereas strong scaling shows consistent speedup and decreasing efficiency. Additionally, both speedup and efficiency are consistently greater for the *Sequential Multi* baseline.



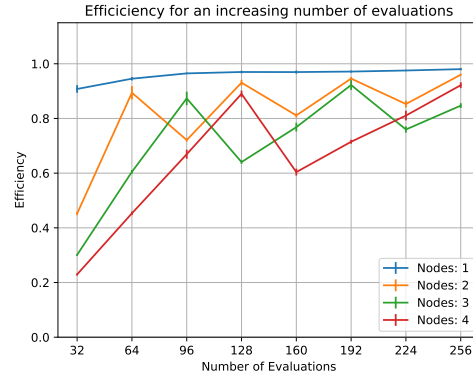
(a) Speedup with *Sequential* baseline



(b) Efficiency with *Sequential* baseline



(c) Speedup with *Sequential Multi* baseline



(d) Efficiency with *Sequential Multi* baseline

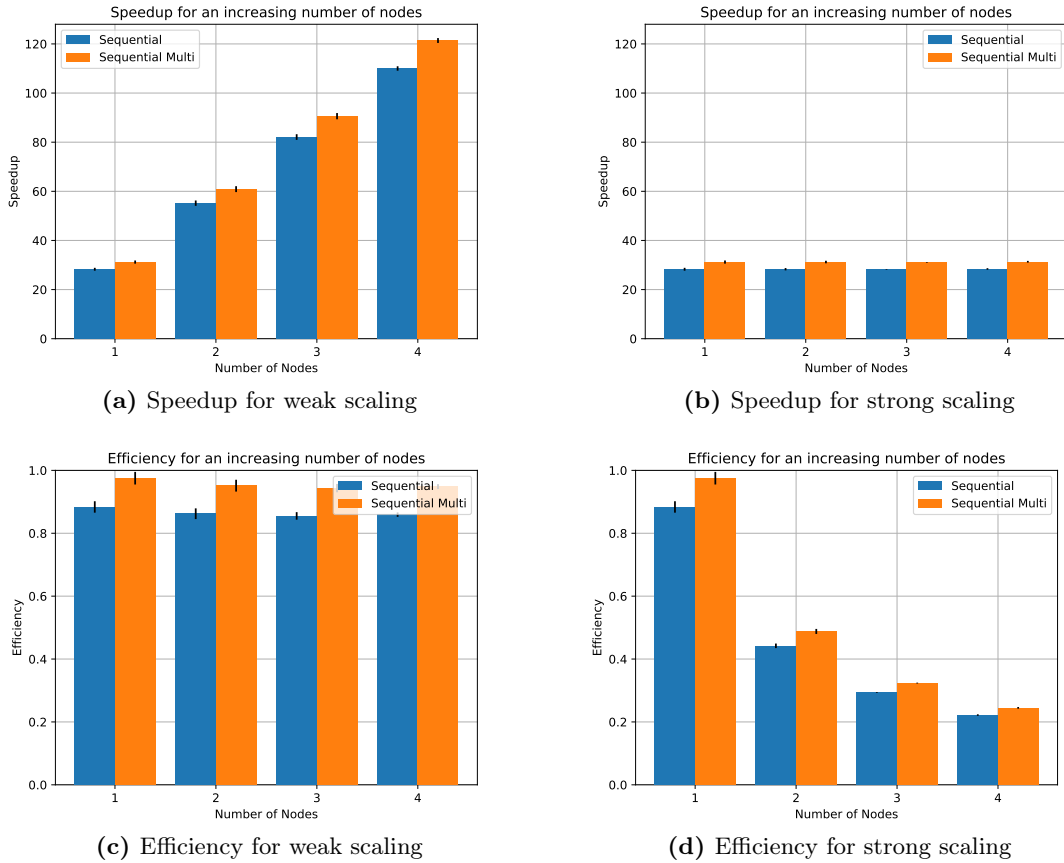
**Figure 5.3:** Efficiency and Speedup for increasing number of simulations with both the *Sequential* and *Sequential Multi* execution time baseline.

Instead of addressing weak and strong scaling for different node configurations, Figure 5.3 captures the speedup and efficiency behaviour of the evaluation environment when increasing the number of evaluations. All four node configurations each exhibit different behaviour. Nevertheless, when increasing the number of evaluations, a general trend upward for both speedup and efficiency is visible. Figures 5.3a and 5.3b represent the speedup



### 5.3 Results

and efficiency when compared against the *Sequential* execution time baseline. Figures 5.3c and 5.3d represent the speedup and efficiency when compared against the *Sequential Multi* execution time baseline. As was also visible in Figure 5.2, when *Sequential Multi* is utilised as baseline, the speedup and efficiency are greater. Figure 5.3d shows that the single node configuration with the *Sequential Multi* baseline approaches an efficiency of 1.0. Dips in both speedup and efficiency are visible for all multi-node configurations. Nevertheless, the trend for multi-node configurations in speedup and efficiency remains upward, where each peak in efficiency or speedup is greater than the previous peak (or at least equal).

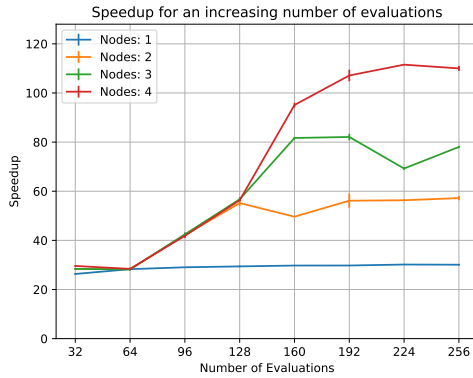


**Figure 5.4:** Efficiency and Speedup for strong and weak scaling with 64 evaluations as baseline for the single node configuration and the *Sequential* and *Sequential Multi* execution time baselines when SMT is enabled.

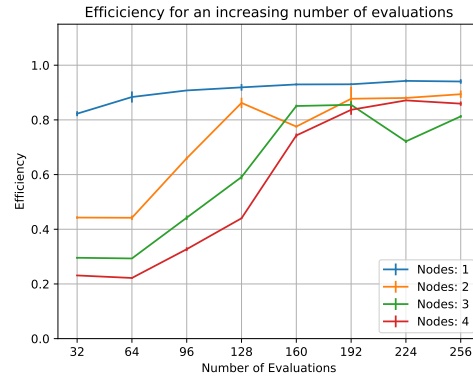
The last part of the simulation campaign experiment investigates the use of SMT. Where the maximum number of evaluations performed concurrently on a single compute node was equal to the number of cores available in a single compute resource for Figures 5.2 and 5.3, the maximum number of evaluations is doubled by supporting the use of SMT in

## 5. EVALUATION

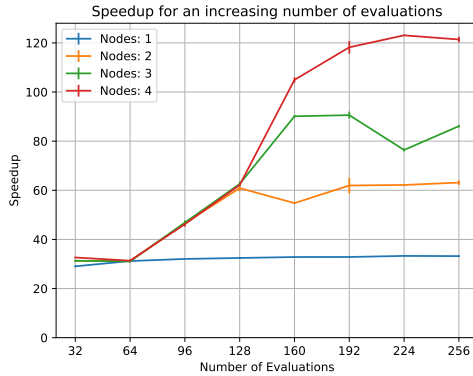
Figures 5.4 and 5.5. The results shown in Figure 5.4 are based on the same weak and strong scaling of the number of nodes experiments as demonstrated in Figure 5.2, but the single node baseline is 64 evaluations instead of 32 evaluations to fill an entire node. As before, weak scaling demonstrates a steady increase in speedup and consistent efficiency, whereas strong scaling shows consistent speedup and decreasing efficiency. The *Sequential Multi* execution time baseline also consistently demonstrates a greater speedup and efficiency when compared against the *Sequential* execution time baseline.



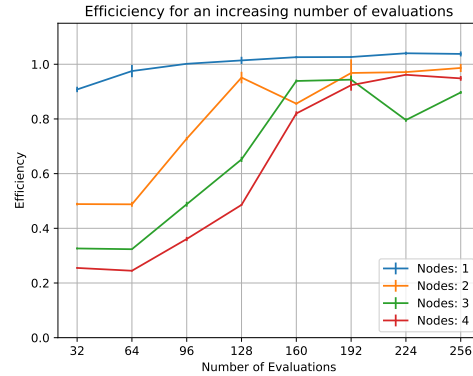
(a) Speedup with *Sequential* baseline



(b) Efficiency with *Sequential* baseline



(c) Speedup with *Sequential Multi* baseline

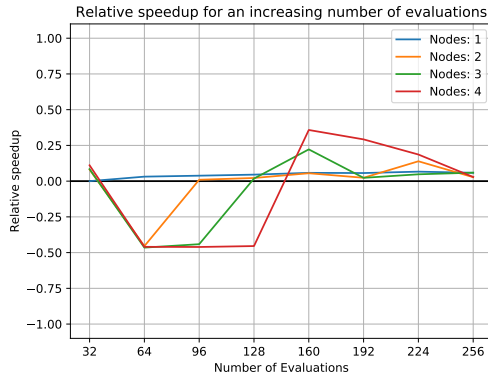


(d) Efficiency with *Sequential Multi* baseline

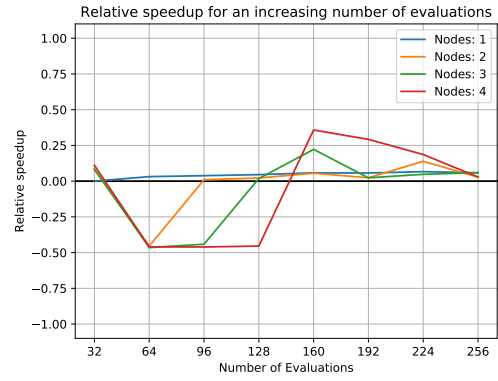
**Figure 5.5:** Efficiency and Speedup for an increasing number of evaluations of the INET-LANS model with both the *Sequential* and *Sequential Multi* execution time baselines when SMT is enabled.

Similarly to Figure 5.3, Figure 5.5 captures the speedup and efficiency behaviour of the evaluation environment when increasing the number of evaluations. A general trend upwards is shown for both speedup and efficiency when increasing the number of evaluations. When *Sequential Multi* is utilised as baseline, the speedup and efficiency are greater, as

was also visible in Figure 5.4. Figure 5.5b shows that the single node configuration with the *Sequential* baseline approaches an efficiency of 1.0. In the case of the *Sequential Multi* baseline, an efficiency greater than 1.0 is achieved. As was the case in Figure 5.3, dips in both speedup and efficiency are visible for all multi-node configurations. Nevertheless, the general trend for multi-node configurations in speedup and efficiency remains upwards, where each peak in efficiency or speedup is greater than the previous peak (or at least equal).



(a) Relative speedup for SMT against non-SMT with *Sequential* baseline and strong scaling ( $S_{rel}(SMT, regular)$ )



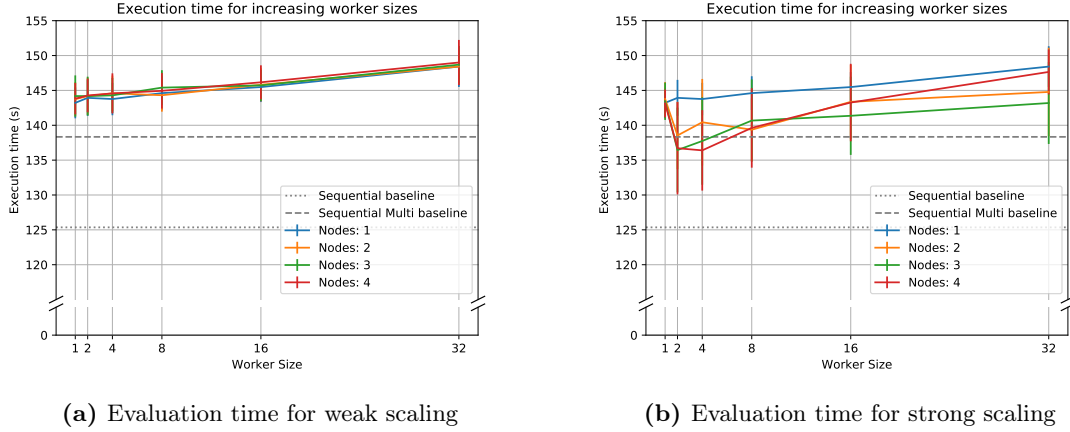
(b) Relative speedup for SMT against non-SMT with *Sequential Multi* baseline and strong scaling ( $S_{rel}(SMT, regular)$ )

**Figure 5.6:** Relative speedup  $S_{rel}$  for an increasing number of evaluations of the INET-LANS model with both the *Sequential*, *Sequential Multi* execution time baselines when SMT is enabled or disabled.

In order to compare the non-SMT and SMT enabled approaches, Figure 5.6 demonstrates the relative speedup  $S_{rel}$  for an increasing number of evaluations with both the *Sequential* and *Sequential Multi* baselines. It is shown that the SMT enabled approach performs, relative to the non-SMT approach, worse with a lower number of evaluations. However, the SMT enabled approach improves and eventually reaches greater performance, relative to the non-SMT approach, for larger number of evaluations. For all number of evaluations considered, the single-node configuration shows improved relative performance when enabling SMT. An increased number of compute nodes also demonstrates a higher number of evaluations before the relative performance has equalised, and afterwards surpassed, for the SMT enabled approach compared to the non-SMT approach.

## 5. EVALUATION

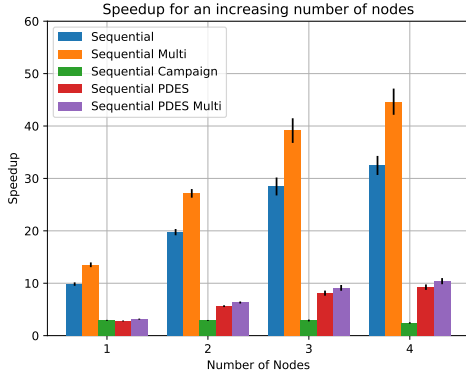
### 5.3.2 Worker Size



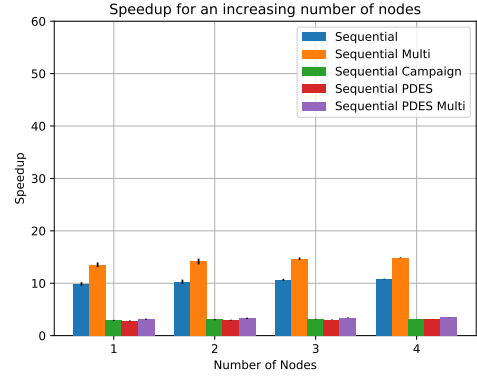
**Figure 5.7:** Evaluation time when increasing the number of workers on a single node for both weak and strong scaling with 32 evaluations as baseline for the single node configuration.

After the simulation campaign experiment follows the worker size experiment, where the goal is to analyse the performance impact of employing different worker configurations in the resource controller. Firstly, data corresponding to the execution time of a single evaluation is displayed in Figure 5.7, where the *Sequential* and *Sequential Multi* execution time baselines are illustrated by the grey dotted horizontal line and striped horizontal line respectively. The baselines are the addition of *Compilation Time* and *Simulation Time*, and the single evaluation time shown is based on the *Worker Time* (as explained in Subsection 5.2.2). Figure 5.7a displays the evaluation time for weak scaling (i.e., for a single node the workload is 32, for two nodes it is 64, for three nodes it is 96, and for four nodes it is 128). Figure 5.7b displays the evaluation time for strong scaling (i.e., for all node configurations, the total workload is 32 evaluations). Overall, an increase in execution time is visible when increasing the number of workers on a single node for all node counts, where strong scaling shows more variability between node configurations.

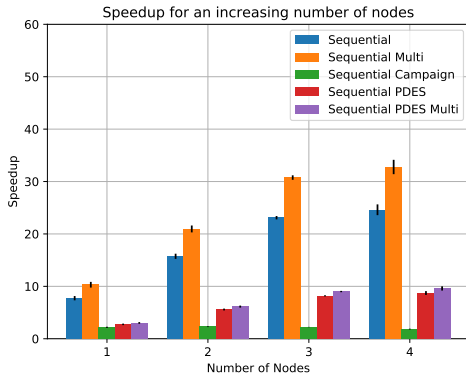
## 5.3.3 PDES



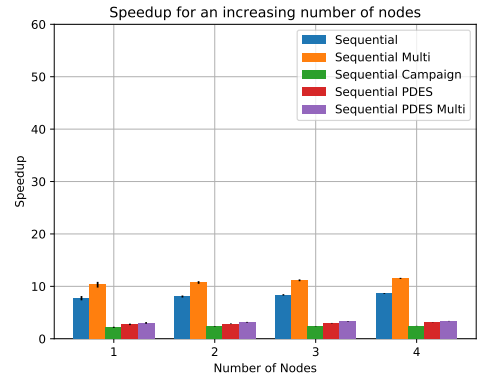
(a) Communication intensive with weak scaling



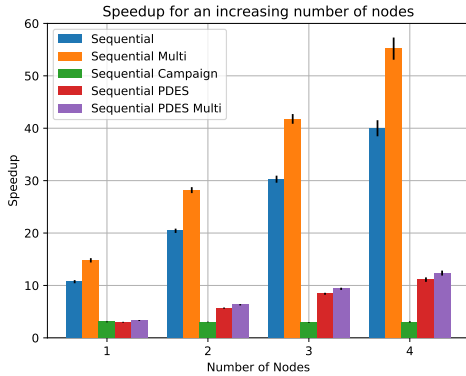
(b) Communication intensive with strong scaling



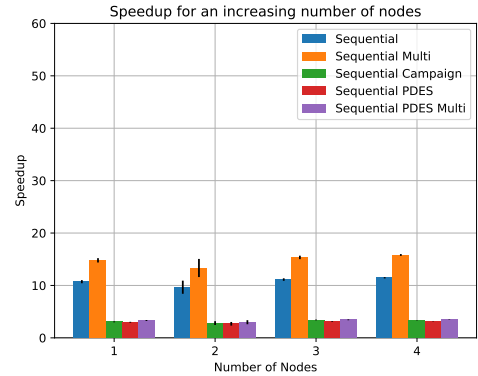
(c) Computation intensive with weak scaling



(d) Computation intensive with strong scaling



(e) Subsystem intensive with weak scaling



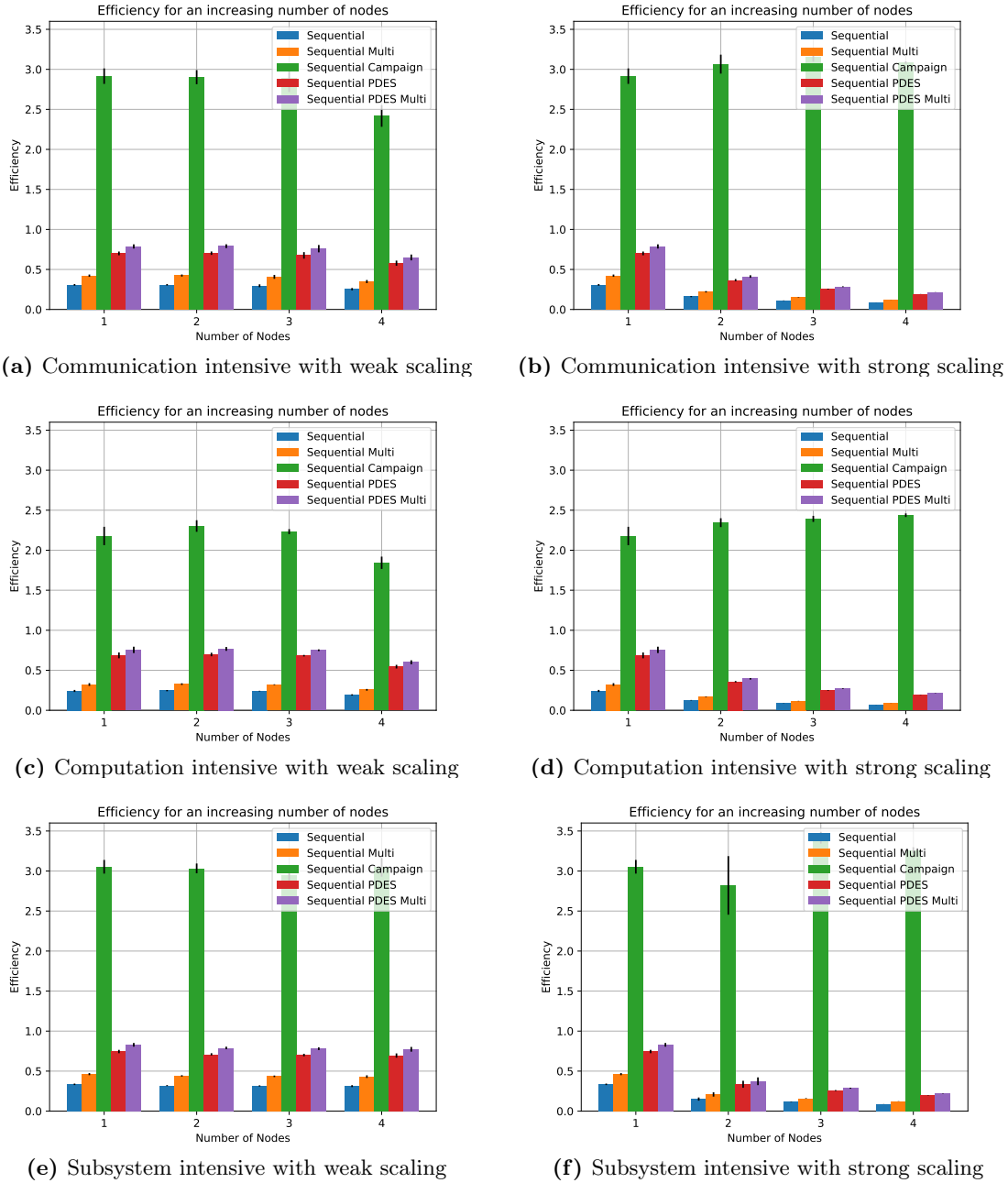
(f) Subsystem intensive with strong scaling

**Figure 5.8:** Speedup for strong and weak scaling with 4 PDES evaluations of a CQN model as baseline for the single node configuration and the *Sequential*, *Sequential Multi*, *Sequential Campaign*, *Sequential PDES*, and *Sequential PDES Multi* execution time baselines.

The final experiment targets the PDES methodology from the proposed solution, whereas the baseline and worker size experiments employed the simulation campaign methodology.

## 5. EVALUATION

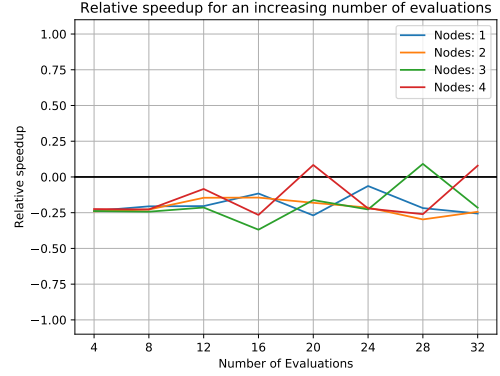
Initially, various baselines are set in Appendix B Figures B.2, B.3, and B.4, which present the performance for the sequential versions of the communication, computation, and subsystem intensive systems with both the *Sequential* and *Sequential Multi* baselines. Similar behaviour is demonstrated, regarding speedup and efficiency for weak and strong scaling of the compute nodes, as was observed in the simulation campaign experiment.



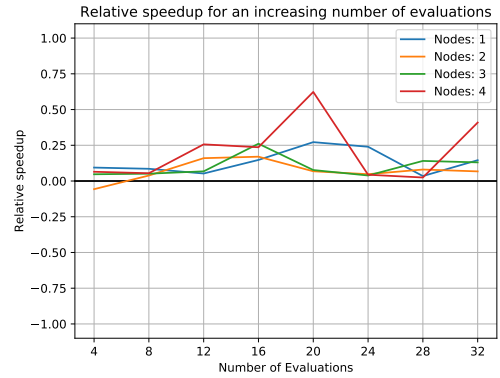
**Figure 5.9:** Efficiency for strong and weak scaling with 4 PDES evaluations of a CQN model as baseline for the single node configuration and the *Sequential*, *Sequential Multi*, *Sequential Campaign*, *Sequential PDES*, and *Sequential PDES Multi* execution time baselines.

Figures 5.8 and 5.9 present the speedup and efficiency for the PDES enabled versions of the communication intensive, computation intensive, and subsystem intensive systems for the *Sequential*, *Sequential Multi*, *Sequential Campaign*, *Sequential PDES*, and *Sequential PDES Multi* execution time baselines. As is shown, the system models demonstrate similar behaviour with regard to speedup and efficiency for weak and strong scaling of the compute nodes as was observed in the simulation campaign experiment. However, both the speedup and efficiency are significantly lower when compared to the *Sequential* and *Sequential Multi* baseline data shown in Figure 5.2. However, for the *Sequential PDES* and *Sequential PDES Multi* baselines, similar performance is shown. The efficiency compared against the *Sequential Campaign* approaches roughly 3.0 for both weak and strong scaling. Between the communication, computation, and subsystem intensive models, similar performance patterns with respect to the different baselines are demonstrated.

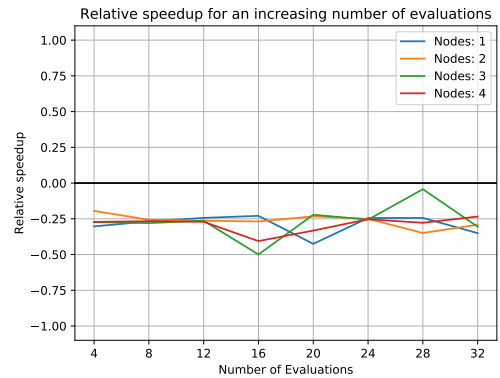
Figures 5.10 and 5.11 show the relative speedup  $S_{rel}$  between the communication intensive, computation intensive, and subsystem intensive models for the *Sequential Multi* and *Sequential PDES Multi* execution time baselines respectively. To concentrate on the impact of the behavioural traits during PDES campaigns compared against a baseline representing full concurrent utilisation of a compute resource,



(a) Computation against communication intensive



(b) Subsystem against communication intensive



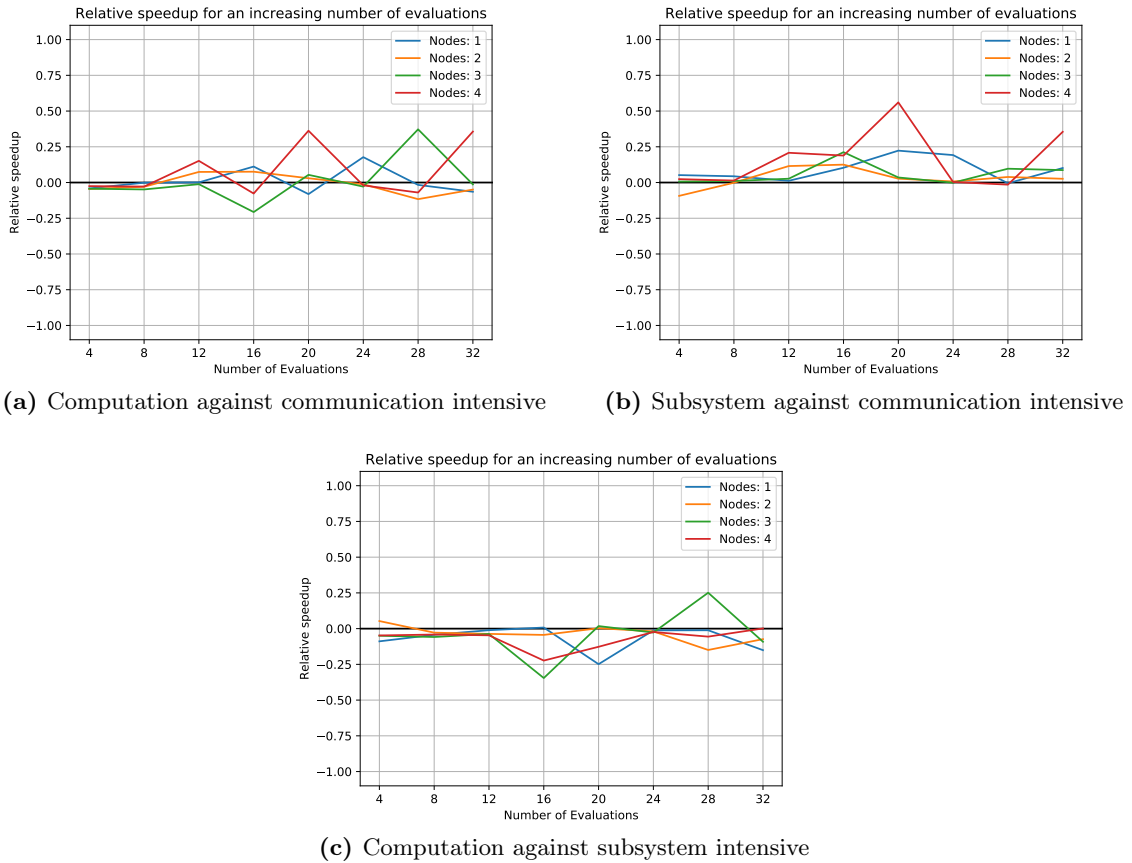
(c) Computation against subsystem intensive

**Figure 5.10:** Relative speedup  $S_{rel}(x, y)$  comparing model  $x$  against model  $y$  for increasing number of PDES CQN model evaluations with the *Sequential Multi* execution time baseline.

## 5. EVALUATION

only relative speedups are shown for the *Sequential Multi* and *Sequential PDES Multi* execution time baselines. All relative speedups  $S_{rel}$ , including the remaining execution time baselines (i.e., *Sequential*, *Sequential Campaign*, and *Sequential PDES Multi*), are demonstrated together in Appendix B.

Figures 5.10 and 5.11 show that the subsystem intensive system model demonstrates a greater relative speedup when compared against the communication and computation intensive models the respective execution time baselines. The communication intensive model shows a greater relative speedup compared to the computation intensive model for the *Sequential Multi* execution time baseline. However, for the *Sequential PDES Multi* execution time baseline, the computation intensive model shows a greater relative speedup compared against the communication intensive model.



**Figure 5.11:** Relative speedup  $S_{rel}(x, y)$  comparing model  $x$  against model  $y$  for increasing number of PDES CQN model evaluations with the *Sequential PDES Multi* execution time baseline.



## 6

# Discussion

After having evaluated the proposed solution, a scalable workflow for an evaluation environment to be utilised in DSE for dCPS, the corresponding outcomes will be analysed and interpreted in this chapter. Specifically, the behaviour, performance, effectiveness, and applicability of the proposed solution will be studied for a variety of use cases. The discussion will cover each experiment, where the evaluation results will be connected and interpreted relative to the research questions, objectives, and context of the research. Finally, an overall reflection is presented on the objectives of the research, its research questions, and the contributions to the research field.

### 6.1 Simulation Campaign

The first experiment defined in the evaluation chapter is the simulation campaign experiment, with the goal to analyse the effectiveness of the solution and utility that can be provided by the proposed environment to DSE for dCPS. Specifically, the simulation campaign methodology is investigated. Figure 5.2 demonstrates, for both speedup and efficiency, the weak and strong scaling behaviour when the INET-LANS model is evaluated in a simulation campaign with 32 evaluations as baseline for the single node configuration. Weak scaling shows a steady proportional growth in speedup when the number of nodes is increased, which translates to a practically consistent efficiency with a standard deviation of 0.02 overall for both execution time baselines. As the efficiency remains consistent, the overhead of scaling the number of compute nodes does not significantly affect the performance of the solution. By providing a steady efficiency when scaling the number of compute nodes, the solution facilitates a predictable service. Strong scaling shows the opposite of weak scaling, where the speedup is practically consistent, which translates to

## 6. DISCUSSION

---

a steady proportional decline in efficiency. The average standard deviation, pertaining to the standard deviations of node configurations, of 0.005 in efficiency for both execution time baselines with strong scaling shows there is less variability in comparison to weak scaling, which has an average standard deviation, pertaining to the standard deviations of node configurations, of 0.015 and 0.017 for the *Sequential* and *Sequential Multi* execution time baselines respectively.

Figure 5.2 also shows that the speedup and efficiency improve when comparing the environment time of the simulation campaign against the *Sequential Multi* baseline instead of the *Sequential* baseline. Due to the *Sequential Multi* baseline utilising all cores on a machine, it is likely that OS interruptions and context switching result in a 10.4% slower execution time baseline. As the *Sequential Multi* baseline is slightly slower in comparison to the *Sequential* baseline, the speedup and efficiency will be slightly greater.

Figure 5.2 only demonstrates the speedup and efficiency for 32 evaluations as baseline for weak and strong scaling on the number of compute nodes. However, the performance and behaviour when scaling the number of evaluations is also of interest, which is presented in Figure 5.3 for both the *Sequential* and *Sequential Multi* baseline. The graphs show strong scaling, as the number of evaluations is consistent between all node configurations. Both the speedup and efficiency graphs demonstrate patterns with drops, which differ for each node configuration. Specifically, the two-node configuration shows drops at 96, 160, and 224 evaluations and peaks at 64, 128, 192, and 256 evaluations in Figures 5.3a, 5.3b, 5.3c, and 5.3d. The pattern of peaks and drops aligns with the multiples of the number of cores available for each node configuration. When the number of evaluations is a multiple of the number of cores available, all nodes will be fully utilised. However, when the number of evaluations is not a multiple of the number of cores available, dividing the evaluations will result in some of the available cores waiting on remaining evaluations to be completed (i.e., there is a  $remainder = evaluations \bmod cores$ , which does not occupy the entirety of the cores available). As such, the environment time of the simulation campaign will be increased by the time of a single evaluation (assuming a perfect transition between evaluations and no additional overhead), but the workload does not grow relatively to the case where the number of evaluations is a multiple of cores. This effect is not seen for the single node configuration, as every sample point is a multiple of its number of cores available.

Figures 5.3b and 5.3d show a growing trend in efficiency when increasing the number of evaluations for all node configurations. A growing trend in efficiency indicates that

the solution benefits from an increased number of evaluations, where the cost of initialisation is masked by an increasing environment time caused by the increment in number of evaluations. Especially, the single node configuration with the *Sequential Multi* baseline approaches an efficiency of 1.0, which indicates that there is little overhead induced by the runtime of the proposed solution (compared against the *Sequential Multi* baseline) and the initial lower efficiency is likely caused by the single-time cost of initialisation. The multi-node configurations also demonstrate a trend towards an efficiency of 1.0 and presumably require more evaluations to close the gap. Additionally, when considering the speedup of multi-node configurations in Figures 5.3a and 5.3c, it is demonstrated that employing more compute nodes on an identical number of evaluations results in a speedup (approximately) equal or greater. Consequently, based on the multi-node configurations evaluated, it can be concluded that the evaluation environment will show improved performance when increasing the number of compute nodes available. However, adding additional compute nodes for improved performance can be considered a trade-off, as it can (eventually) show a diminishing return on investment. For example, when evaluating 128 system models, Figure 5.3a shows a slight improvement when employing three compute nodes over two compute nodes. However, this slight increase in performance might not be worth it to the designer or system administrator to employ an additional compute node. Furthermore, when the evaluations become thinly spread over the compute nodes (or when there are more compute nodes than evaluations), adding an additional compute node might introduce more overhead than it will provide additional performance to the system.

Lastly, the impact of Simultaneous MultiThreading (SMT) was analysed. Figure 5.4 demonstrates the weak and strong scaling behaviour when enabling SMT, with 64 evaluations as the baseline for a single node configuration. Comparing the data to the non-SMT approach outcomes shown in Figure 5.2 demonstrates similar behaviour with an overall slightly higher speedup and efficiency. SMT could therefore have better utilisation of the compute node resources. However, the improvement could also be caused by the increased number of evaluations, which can improve the overall speedup and efficiency, as was shown for the non-SMT approach data in Figure 5.3. Figure 5.5 demonstrates the speedup and efficiency when scaling the number of evaluations with SMT-enabled. When comparing Figures 5.3 (non-SMT) and 5.5 (SMT-enabled) to analyse the behaviour and impact of enabling SMT, it is observed that for lower number of evaluations the speedup and efficiency are significantly lower when enabling SMT (i.e., a 44.1% decrease from non-SMT to SMT enabled for the three nodes with 96 evaluations configuration). This initial phase of worse performance is caused by the division of evaluations over the compute nodes.

## 6. DISCUSSION

---

As each node now supports 64 evaluations, performing 64 evaluations on two nodes only utilises all 64 available spots on a single node. If the workload division in the initial phase would be distributed more equally, the performance would likely equal the non-SMT approach. Nevertheless, once the full capacity of the compute node(s) has been reached, the SMT-enabled approach outperforms the non-SMT approach for the same number of evaluations (i.e., a 3% increase from non-SMT to SMT enabled for the four nodes with 256 evaluations configuration and 12.2% increase from non-SMT to SMT enabled for the two nodes with 224 evaluations configuration). Additionally, the pattern of drops will be of lower frequency, as the tipping points require more evaluations. Where the single node configuration approached 1.0 and the multi-node configurations demonstrated a similar trend in Figure 5.3d, when enabling SMT, the single node configuration beats the 1.0 efficiency and the two- and three-node configurations approach 1.0. Figure 5.6 demonstrates the relative speedup between the SMT-enabled and non-SMT versions, showing the advantage of non-SMT at a lower number of evaluations, and SMT providing greater speedups for a larger number of evaluations. By enabling SMT, the theoretical peak performance, which considers the full utilisation of a single core by a single evaluation based on the respective execution time baseline, is beaten. Thereby demonstrating that enabling SMT significantly improves the utilisation of a compute node, providing an increase in performance compared to the non-SMT approach for both the *Sequential* and *Sequential Multi* execution time baselines.

When considering Research Question 1.1 on enabling efficient and scalable evaluation of the vast design space of complex, distributed Cyber-Physical Systems, the proposed solution shows it can provide efficient and scalable evaluation when employing the simulation campaign methodology with the complex INET-LANS model mimicking a dCPS. Both experiments on scalability in compute nodes and scalability in the number of evaluations demonstrate the capability of the proposed solution to facilitate scalable and efficient evaluation when scaling the number of compute nodes and number of evaluations. When SMT is enabled, the efficiency of the simulation campaign methodology increases further and the environment achieves a greater utilisation of the available compute nodes.

### 6.2 Worker Size

The second experiment defined in the evaluation chapter is the worker size experiment, with the goal to analyse the performance impact of employing different worker configurations in the resource controller. Similarly to the simulation campaign experiment, the

simulation campaign methodology is investigated together with the INET-LANS model. However, instead of investigating the performance and behaviour when scaling the number of compute nodes or number of evaluations, the impact of the number of workers present on a single compute node is analysed. The number of workers indicates the number of node-local task manager processes, where each worker can manage multiple tasks (i.e., a single worker could manage 32 evaluations, or four workers could each manage eight evaluations).

Figures 5.7a and 5.7b show the impact of scaling the number of workers on a single node for both weak and strong scaling, with 32 evaluations as baseline for the single node configuration. Both the weak scaling and strong scaling graphs demonstrate an upwards trend in evaluation time, where an increase in worker size yields a slowdown in evaluation time. Where weak scaling shows consistent behaviour between node configurations, strong scaling shows more variability between node configurations. This difference is likely caused by the division of work, causing reduced workloads per additional compute node added, which is inherent to strong scaling. The upwards trend in evaluation time indicates that the increase in worker size reduces the performance of the solution, likely caused by the overhead of additional processes present on the compute nodes. At the scale of the number of compute nodes evaluated, there does not seem to be a significant impact on the evaluation time when scaling the number of compute nodes (and the number of worker processes per compute node remains the same) for weak scaling.

Besides the increase in evaluation time, the increase in number of workers could also degrade the performance of the internal Dask Scheduler, which is responsible for the internal communication. An increased number of workers will result in more communication, synchronisation, and administration overhead in the resource controller. However, this is likely to only occur with a significantly larger number of compute nodes. Nevertheless, reducing the number of workers required per compute node will reduce the likelihood of encountering this problem.

The choice for the number of workers can depend on multiple factors, such as redundancy and global locks. In the case of simulations through OMNeT++, it is shown that a single worker suffices. The reason a single worker suffices is likely caused by the Python GIL (Global Interpreter Lock) [62]. As the OMNeT++ process launched by a task in the worker process does not occupy the GIL, a worker can manage multiple evaluations in parallel. Whenever tasks would require the GIL, multiple worker processes would be advisable in order to reduce contention (i.e., if all tasks require the GIL, a single worker per task will likely be appropriate).

## 6. DISCUSSION

---

When considering Research Question 1.1 on enabling efficient and scalable evaluation of the vast design space of complex, distributed Cyber-Physical Systems, the worker size experiment shows, when employing the proposed solution with the INET-LANS model mimicking a dCPS, scalable and efficient evaluation is best achieved when applying a single worker process per compute node. However, considering Research Question 1, different workloads utilising the same workflow might require a different configuration.

### 6.3 PDES

The last experiment defined in the evaluation chapter is the PDES experiment, with the goal of investigating the performance impact of different behavioural traits in system models and gaining insights into the applicability of the PDES methodology for different kinds of systems. Where the simulation campaign and worker size experiments concerned the simulation campaign methodology, the DSE process might prefer (or require) the PDES methodology over the simulation campaign methodology, depending on the system models to be explored. Compared against the simulation campaign experiment, the sequential versions of the communication, computation, and subsystem intensive CQN models demonstrate similar behaviour pertaining to the speedup and efficiency of weak and strong scaling (see Figures B.2, B.3, and B.4 respectively).

The speedup and efficiency achieved by performing a simulation campaign with four instances of the PDES-enabled CQN models as baseline for the single node configuration are shown in Figures 5.8 and 5.9. As was the case for their respective sequential models, the behaviour pertaining to weak and strong scaling of compute nodes is similar to the simulation campaign experiment. Additionally, the different models demonstrate roughly similar performance. However, the subsystem intensive CQN model has a slightly higher speedup and efficiency overall. The efficiency of the PDES campaign compared to the *Sequential* and *Sequential Multi* execution time baselines is relatively low (below 0.5) for all CQN models. Nevertheless, when compared against the *Sequential PDES* and *Sequential PDES Multi* execution time baselines, the performance approaches the efficiency as was seen in the simulation campaign experiment. Therefore, the integration of PDES-enabled system models in a simulation campaign does not significantly alter the behaviour and performance relative to a sequential system model execution campaign, which is compared against a sequential system model execution baseline (i.e., the *Sequential* and *Sequential Multi* baselines). An outlier in efficiency is demonstrated when considering the *Sequential Campaign* execution time baseline. As the simulation campaign on four sequential versions

of the corresponding CQN model is highly inefficient considering the utilisation of the available compute capacity, the efficiency of four PDES-enabled CQN models utilising the entire compute node compared against the inefficient sequential simulation campaign is significantly above 1.0 (i.e., 2.78, 2.14, and 3.02 average efficiency for the communication, computation, and subsystem intensive model respectively with weak scaling).

In order to compare the communication, computation, and subsystem intensive CQN models, Figures 5.10 and 5.11 demonstrate the relative speedup  $S_{rel}$  for the *Sequential Multi* and *Sequential PDES Multi* execution time baselines respectively. The relative speedups for the *Sequential*, *Sequential Campaign*, and *Sequential PDES* execution time baselines can be found in Figures B.5, B.7, and B.8 respectively. The relative speedup  $S_{rel}$  enables a clear comparison of the three models, each representing a different behavioural trait, for the different execution time baselines. It is demonstrated that the subsystem intensive CQN model has a greater relative speedup compared to both the communication and computation intensive models for all execution time baselines. Comparing the communication and computation intensive CQN models shows that the communication intensive system has greater relative speedup for the *Sequential*, *Sequential Multi*, and *Sequential Campaign* baselines. However, for the *Sequential PDES* and *Sequential PDES Multi* baselines, the computation intensive model surpasses the communication intensive model. Therefore, the communication intensive model demonstrates a greater relative speedup for all sequential system execution baselines, whereas the computation intensive model exceeds the communication intensive model for all parallel system execution baselines. One possible explanation for this behaviour is that decomposing a computational intensive model adds (relatively) more overhead, thereby creating a relatively slower PDES-based execution time baseline, than is the case for a communication intensive model. However, a more in-depth and extensive analysis would be required to answer conclusively.

Overall, all three CQN PDES system models exhibit similar performance behaviour related to weak and strong scaling of the number of compute nodes and when scaling the number of evaluations. The subsystem intensive CQN model slightly outperforms the communication intensive and computation intensive models. As the PDES implementation in OMNeT++ utilises conservative synchronisation, the differences in performance of the three different behavioural traits might be more pronounced when employing different, more optimistic synchronisation protocols.

When considering Research Question 1.1 on enabling efficient and scalable evaluation of the vast design space of complex, distributed Cyber-Physical Systems, the proposed solution shows it can provide efficient and scalable evaluation when employing the PDES

## 6. DISCUSSION

---

methodology with a variety of PDES-enabled CQN models. A difference in performance between system models, each exhibiting a distinct behavioural trait, was demonstrated. The difference in performance poses a key consideration (Research Question 1) when utilising the proposed solution, as the characteristics of a system could influence the choice between sequential evaluation or PDES-enabled evaluation and the decomposition of the model during the DSE process.

### 6.4 General

The objective of this research, set in Chapter 1, was to address the challenge of scalability in the evaluation environment of the design process employed by researchers and designers of the next-generation dCPS. Thereby investigating and developing a scalable and efficient approach that can evaluate a vast number of complex dCPS design points by exploiting distributed simulation techniques. In order to address the objectives, a main research question and subsequent research questions extending it, were formulated. The main Research Question 1.1 investigated how a distributed evaluation workflow can be designed and dynamically adapted to enable efficient and scalable evaluation of the vast design space of complex, distributed Cyber-Physical Systems. In this thesis, an evaluation environment was proposed to facilitate scalable and efficient evaluation in the context of DSE for dCPS. Throughout the approach and methodology, the current state-of-the-art was considered (and applied where necessary), whilst also considering innovative, novel approaches and taking into account requirements and desirable features with regards to DSE for dCPS. Identifying and exploring design considerations to develop a distributed evaluation workflow for efficient and scalable evaluation of complex, distributed cyber-physical systems corresponds to Research Question 1. By inspecting, utilising, evaluating, and combining the various design possibilities, a complete evaluation environment was proposed, implemented, and evaluated. Through the evaluation, Research Question 1 was addressed, and the solution demonstrated a distributed evaluation workflow designed to scale evaluations efficiently across multiple computing resources. The considerable breadth of design considerations, extensive evaluation, and significant context dependant implementation requirements discovered during the study of the state-of-the-art, design proposal, and implementation of the workflow, have contributed to a variety of possible future research directions for improving distributed evaluation workflows and addressing scalability challenges in the design space exploration of complex, distributed cyber-physical systems.



Besides investigating the efficiency and scalability, the solution has a variety of features and facilities to enable and cater to a diverse set of DSE for dCPS environments. Firstly, the hierarchical implementation of design point queues, where each queue is regulated by its own scheduling policy, allows the designer to configure dynamic input handling, enabling complex interactions between the evaluation environment and the search algorithm (or other external applications). Another feature of the evaluation environment, enabling it to cater to a variety of deployments for DSE of dCPS, is that its agnostic to the underlying simulator infrastructure. The evaluation environment utilises a DPO to communicate the design points between workflow components, where only a subset (i.e., general data like the UID and local storage location) of the DPO parameters are accessed by the environment. Inside each DPO is defined where the design point data is stored and how it should compile and execute the respective design point. Other simulator frameworks can be supported inside the evaluation framework by configuring the compilation and execution functionality of the DPO according to the respective simulator. Together with the DPO, the task distributed by the Dask framework can be adjusted to integrate more functionality required (currently, it only calls the compile and execution functionality of the DPO).

Another area where the evaluation environment can be adapted is the compute infrastructure. The configuration of the environment parses the compute infrastructure settings and forwards them accordingly to Dask, enabling the solution to easily be scaled up and down (from simple tests on a laptop to full-scale deployment on a large cluster) and utilise a variety of commonly used CMS like PBS, SLURM, LSF, and SGE. Facilitating small-scale tests and a variety of common infrastructures allows the evaluation environment to be adopted and utilised in a broad range of environments targeting DSE for dCPS or similar applications.

During the approach, methodology, and evaluation of the proposed evaluation environment, simulation campaigns and PDES were the two main scalability methodologies discussed. By facilitating both workload-level and task-level parallelism, the solution supports the utilisation of the two methodologies both independently and simultaneously. The support of the simulation campaign is inherently available through the distribution of the workload over compute resources. Task-level parallelism is enabled through the slot-mechanic, described in Subsection 4.2.3, supporting a variety of paradigms (i.e., PDES and OMNeT++ native parameter studies). Additionally, the slots mechanic enables the inclusion of custom task-level approaches, as the task-level parallelism is encoded in the DPO simulation execution definition and not accessed by the workflow components. Thereby, not only are the PDES paradigm and the OMNeT++ native parameter study supported,

## 6. DISCUSSION

---

future task-level parallelism execution of other simulator frameworks or DSE-related applications are facilitated.

## Conclusion

This thesis considers the entirety of the evaluation environment to address the scalability challenge in the design process employed by researchers and designers of the next-generation complex dCPS. The research builds on a prior literature study investigating state-of-the-art methods and techniques for the scalability of system-level simulation environments applicable to the evaluation of dCPS designs [30]. Both the literature study and this thesis were conducted in the context of the DSE2.0 project [24], where the entirety of the state-of-the-art in Design Space Exploration is under scrutiny. Two challenges in advancing the field towards efficient and scalable DSE for distributed Cyber-Physical Systems were identified by the DSE2.0 project; (i) Modelling complex dCPS, and (ii) Scalable DSE.

Especially challenge (ii) was of interest to the research in this thesis. In order to address the scalability challenge in DSE of complex dCPS, the research started by creating a baseline of understanding of the state-of-the-art, its complexity, research directions, and challenges. Research areas such as CPS, DSE, Scalability, and Distributed Computing were considered. A related works section discussed the state-of-the-art methods and techniques researching scalability of system-level simulation environments applicable to DSE of dCPS. The capability and applicability of existing scalability and DSE methodologies were investigated, where it was argued that existing approaches do not sufficiently address the challenge of scalability in the context of DSE for complex dCPS.

After having investigated, reviewed, and discussed the background and related works pertaining to the challenge of scalability in the evaluation environment for the DSE of dCPS, a solution was proposed in the approach and methodology. The proposed evaluation environment aims to provide scalability by distributing the workload over compute resources in a

## 7. CONCLUSION

---

compute cluster and enabling the search algorithm (or other external applications) to submit design points to be evaluated, where a hierarchical input queue can adapt to facilitate dynamic, complex interactions. The environment simultaneously supports workload-level and task-level parallelism by distributing evaluations (simulation campaigns) and facilitating task-level parallelism paradigms (i.e., PDES and parameter studies). Evaluation of design points is simulator agnostic, enabling the solution to be adopted in a variety of DSE environments. Furthermore, integration of the solution on a diverse set of compute infrastructures is supported by the resource controller, which can be configured to utilise a variety of well-known Cluster Management Software environments (e.g., SLURM, PBS, and LSF). Overall, the proposed evaluation environment provides a scalable and efficient workflow, facilitating complex and dynamic interactions with external applications while also providing the freedom of adapting and integrating the solution with a diverse set of infrastructures, environments, and use cases.

The evaluation of the proposed solution investigated the performance and behaviour of the simulation campaign scalability methodology, the configuration of the resource controller, and the influence of different system characteristics on the effectiveness of the PDES scalability methodology. Throughout the evaluation, a variety of system models and execution time baselines (i.e., single sequential evaluation and multiple sequential evaluations) were used to calculate the corresponding speedup and efficiency. A large Ethernet campus backbone model, containing around 8000 computers and 900 switches and hubs, was utilised to represent a design point during DSE of dCPS for both the simulation campaign scalability methodology and the configuration of the resource controller evaluations. Analysing the results of the baseline simulation campaign evaluation showed consistent weak scaling behaviour (pertaining to the scaling of compute nodes) and an efficiency trend towards 1.0 when scaling the number of evaluations, demonstrating a scalable and efficient solution by distributing the workload using simulation campaigns. Additionally, when enabling SMT, compute resource utilisation grows when scaling the number of evaluations, amplifying the efficiency to slightly above 1.0, demonstrating an improved utilisation compared against the full utilisation of the compute resource by the respective execution time baseline. During the evaluation of the resource controller configuration, the impact of the number of worker processes present on a single compute node is evaluated. The results showed that for the simulation workload employed in this research, a single worker suffices, and increasing the number of workers per compute node amounts to an increased evaluation execution time. In the PDES methodology evaluation, three different Closed Queueing Network (CQN) system models were utilised to analyse the behaviour and performance impact of a system

---

its behavioural characteristics, where a single model represented either a communication, computation, or subsystem intensive system. All sequential versions of the CQN models demonstrated similar performance and behaviour as was shown during the baseline simulation campaign evaluation. When comparing the campaigns of PDES-enabled system models against the PDES-enabled execution time baselines, similar (although slightly lower) performance and behaviour are demonstrated pertaining to the weak and strong scaling of compute nodes when compared against the respective model its sequential simulation campaigns. However, when comparing the PDES-enabled simulation campaign to the sequential execution time baselines, the efficiency does not surpass 0.5. A lower efficiency indicates a reduced, less-optimal utilisation of the available compute resources when enabling PDES compared against sequential evaluation. Analysing the performance and behaviour of the three different behavioural traits, it was shown that the subsystem intensive system model outperforms, based on relative speedup, the communication and computation intensive models for all execution time baselines. Between the communication and computation intensive models, the relative performance depends on the execution time baseline. As it is a case study utilising three system models, the observations can not be generalised to a conclusion for all systems or behavioural traits. The efficiency and utility of enabling PDES will, therefore, depend on the system to be evaluated, the use case, and the DSE process. Nevertheless, it is shown that enabling PDES can provide improved evaluation performance, and a system model its behavioural characteristics could influence the performance of its evaluation.

Based on the evaluation of the proposed solution, the simulation campaign with a single worker process per compute node would provide a suitable baseline for scalability in the context of DSE2.0, providing high efficiency when scaling the number of evaluations and enabling SMT. The PDES methodology, or other task-level parallelism paradigms, demonstrate improved single evaluation performance, but have lower simulation campaign efficiencies. Utilising such paradigms effectively will, therefore, depend on the use case or system model itself (i.e., a large system model might require additional resources to be processed effectively). Nevertheless, the proposed solution enables the designer to combine both simulation campaigns and PDES, allowing system models that require the PDES methodology to utilise it and other systems models to remain sequential evaluations in the simulation campaign. Furthermore, the solution facilitates the adaptation and integration into a variety of computing infrastructures, external applications, and use cases by enabling the configuration and customisation of task-level parallelism paradigms, simulator execution and frameworks, dynamic-input handling, and Cluster Management Software.

## 7. CONCLUSION

---

Overall, the research presents a design, implementation, and evaluation of a distributed evaluation workflow dynamically adaptable to its use case and infrastructure, enabling efficient and scalable evaluation of a vast collection of system models to address the challenge of scalable and efficient evaluation of the vast design space of complex, distributed Cyber-Physical Systems.

### 7.1 Future Work

Throughout the research, analysis, and implementation efforts to investigate and realise a scalable and efficient evaluation environment for DSE of dCPS in the context of DSE2.0, various areas of future research were identified that can provide insights to further the facilitation of scalable design space exploration for dCPS. Firstly, the evaluation of the proposed solution demonstrated promising results pertaining to the behaviour and performance when scaling the number of evaluations, the number of compute resources, or both. Nevertheless, a higher statistical accuracy could be achieved with more extensive evaluation (i.e., performing additional executions of the experiments). Moreover, additional experiments with different characteristics could be performed to further investigate and understand the behaviour and performance of the proposed evaluation environment. For instance, analyse the behaviour when employed by an actual search algorithm in the context of DSE, or analyse the impact of large quantities of compute resources (and thereby worker processes) on the efficiency of the solution.

An area where additional research could be performed is the internal components of the proposed workflow. A hierarchical set of design point queues are available inside the proposed workflow, facilitating complex interactions between the search algorithm (or any other external application) and the resource controller. However, the set of design point queues is not aware of the current state of the compute resources. Advanced integration of the design point queues could enable resource-aware scheduling policies, which can assess the state of the environment and schedule design points more intelligently (i.e., schedule to more energy-efficient resources when possible or try to distribute the workload more evenly based on real-time hardware performance). An additional research direction could investigate complex and advanced similarity analysis of the design points to improve its efficacy significantly. Another area in the workflow that can be investigated further is the interaction between the workflow components. Currently, only the resource controller can process the design points asynchronously. When all workflow components would function asynchronously, more complex workflows could be supported. For instance, instead of the

search algorithm initiating the evaluation of design points, the resource controller could continuously request design points in case compute resources become available.

Currently, the evaluation environment contains a single resource controller facilitating the distribution of the workload and managing the available compute resources. However, supporting a collection of resource controllers could enable a more complex interaction between the search algorithm, workload, and compute resources. A potential use case for additional resource controllers could be to pair them with a set of design point queues, allowing the evaluation environment to implement priority-based scheduling and execution or to introduce more complex workflows. Furthermore, separate resource controllers could enable the evaluation environment to interface with multiple compute clusters, each implementing its own CMS.

Another area of interest in the resource controller could be to investigate task-level parallelism on multiple nodes. Currently, task-level parallelism (e.g., PDES) can only be performed within a single compute node. By supporting multi-node task-level parallelism, more complex evaluations or task-level parallelism paradigms can be supported.

Considering the PDES methodology, the evaluation thereof utilised a set of system models as a case study. Further research could investigate the impact of a system its behavioural characteristics in more detail, potentially reaching a verdict generalisable to dCPS system models. During the execution of a PDES-enabled system model, the decomposition of the model into its logical processes has to manually be performed beforehand (i.e., the model is not automatically decomposed into a set of independent processes by OMNeT++). Together with further research on behavioural characteristics, another area of interest could be to automatically (optimally) decompose system models based on property analysis.





# Bibliography

- [1] *520.omnetpp\_r*. URL: [https://www.spec.org/cpu2017/Docs/benchmarks/520.omnetpp\\_r.html](https://www.spec.org/cpu2017/Docs/benchmarks/520.omnetpp_r.html) (visited on 06/20/2023).
- [2] Benny Akesson, Mitra Nasri, Geoffrey Nelissen, Sebastian Altmeyer, and Robert I. Davis. “A comprehensive survey of industry practice in real-time systems”. In: *Real-Time Systems* (2021). Publisher: Springer, pp. 1–41.
- [3] Gene M. Amdahl. “Validity of the single processor approach to achieving large scale computing capabilities”. In: *Proceedings of the April 18-20, 1967, spring joint computer conference*. 1967, pp. 483–485. DOI: 10.1145/1465482.1465560.
- [4] Radhakisan Baheti and Helen Gill. “Cyber-physical systems”. In: *The impact of control technology* 12.1 (2011), pp. 161–166.
- [5] Henri Bal, D. Epema, Cees Laat, Rob Van Nieuwpoort, John Romein, Frank Seinstra, Cees Snoek, and Harry Wijshoff. “A Medium-Scale Distributed System for Computer Science Research: Infrastructure for the Long Term”. In: *IEEE Computer* 49 (May 1, 2016), pp. 54–63. DOI: 10.1109/MC.2016.127.
- [6] Daniel Balouek, Alexandra Amarie, Ghislain Charrier, Frédéric Desprez, Emmanuel Jeannot, Emmanuel Jeanvoine, Adrien lèbre, David Margery, Nicolas Niclausse, Lucas Nussbaum, Olivier Richard, Christian Pérez, Flavien Quesnel, Cyril Rohr, and Luc Sarzyniec. *Adding Virtualization Capabilities to Grid’5000*. Vol. 367. Journal Abbreviation: Communications in Computer and Information Science Publication Title: Communications in Computer and Information Science. July 31, 2012. ISBN: 978-3-319-04518-4. DOI: 10.1007/978-3-319-04519-1\_1.
- [7] Pablo Andrés Barbecho Bautista, Luis Felipe Urquiza-Aguilar, Leticia Lemus Cárdenas, and Mónica Aguilar Igartua. “Large-scale simulations manager tool for OM-NeT++: expediting simulations and post-processing analysis”. In: *IEEE access* 8 (2020). Publisher: IEEE, pp. 159291–159306. DOI: 10.1109/ACCESS.2020.3020745.

## BIBLIOGRAPHY

---

- [8] Adam L. Beberg, Daniel L. Ensign, Guha Jayachandran, Siraj Khaliq, and Vijay S. Pande. “Folding@ home: Lessons from eight years of volunteer distributed computing”. In: *2009 IEEE International Symposium on Parallel & Distributed Processing*. IEEE, 2009, pp. 1–8. DOI: 10.1109/IPDPS.2009.5160922.
- [9] Behzad Boroujerdian, Ying Jing, Devashree Tripathy, Amit Kumar, Lavanya Subramanian, Luke Yen, Vincent Lee, Vivek Venkatesan, Amit Jindal, Robert Shearer, and Vijay Janapa Reddi. “FARSI: An Early-stage Design Space Exploration Framework to Tame the Domain-specific System-on-chip Complexity”. In: *ACM Transactions on Embedded Computing Systems* 22.2 (Jan. 24, 2023), 31:1–31:35. ISSN: 1539-9087. DOI: 10.1145/3544016. URL: <https://dl.acm.org/doi/10.1145/3544016> (visited on 03/21/2023).
- [10] Anastasiia Butko, Rafael Garibotti, Luciano Ost, and Gilles Sassatelli. “Accuracy evaluation of gem5 simulator system”. In: *7th International workshop on reconfigurable and communication-centric systems-on-chip (ReCoSoC)*. IEEE, 2012, pp. 1–7. DOI: 10.1109/ReCoSoC.2012.6322869.
- [11] Lukai Cai and Daniel Gajski. “Transaction level modeling: an overview”. In: *First IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and Systems Synthesis (IEEE Cat. No. 03TH8721)*. IEEE, 2003, pp. 19–24.
- [12] Celery - Distributed Task Queue — Celery 5.3.1 documentation. URL: <https://docs.celeryq.dev/en/stable/> (visited on 02/10/2023).
- [13] DAS-6: Distributed ASCI Supercomputer 6. URL: <https://www.cs.vu.nl/das/> (visited on 07/03/2023).
- [14] Dask — Dask documentation. URL: <https://docs.dask.org/en/stable/> (visited on 06/20/2023).
- [15] Dask | Scale the Python tools you love. URL: <https://www.dask.org/> (visited on 06/20/2023).
- [16] Dask-Jobqueue — Dask-jobqueue 0.8.2+0.gff47d71.dirty documentation. URL: <https://jobqueue.dask.org/en/latest/> (visited on 06/20/2023).
- [17] Dask.distributed — Dask.distributed 2023.6.0 documentation. URL: <https://distributed.dask.org/en/stable/> (visited on 06/20/2023).
- [18] Thomas Dreibholz. “Evaluation and Optimisation of Multi-Path Transport using the Stream Control Transmission Protocol”. PhD thesis. Mar. 13, 2012.

## BIBLIOGRAPHY

---

- [19] Greg Ewing, Krzysztof Pawlikowski, and Don McNickle. “Akaroa-2: Exploiting network computing by distributing stochastic simulation”. In: (1999). Publisher: SCSIPress.
- [20] Julius Flohr. *Neurogenesis*. original-date: 2016-07-20T16:02:39Z. July 21, 2021. URL: <https://github.com/juliusf/Neurogenesis> (visited on 04/10/2023).
- [21] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. “Cloud computing and grid computing 360-degree compared”. In: *2008 grid computing environments workshop*. Ieee, 2008, pp. 1–10. DOI: 10.1109/GCE.2008.4738445.
- [22] GEM5. *gem5: About*. URL: <https://www.gem5.org/about/> (visited on 11/23/2023).
- [23] John L. Gustafson. “Reevaluating Amdahl’s law”. In: *Communications of the ACM* 31.5 (1988). Publisher: ACM New York, NY, USA, pp. 532–533. DOI: 10.1145/42411.42415.
- [24] Marius Herget, Faezeh Sadat Saadatmand, Martin Bor, Ignacio González Alonso, Todor Stefanov, Benny Akesson, and Andy D. Pimentel. “Design Space Exploration for Distributed Cyber-Physical Systems: State-of-the-art, Challenges, and Directions”. In: 2022 25th Euromicro Conference on Digital System Design (DSD). IEEE, 2022, pp. 632–640. ISBN: 978-1-66547-404-7. DOI: 10.1109/DSD57027.2022.00090.
- [25] Mark D. Hill. “What is scalability?” In: *ACM SIGARCH Computer Architecture News* 18.4 (1990). Publisher: ACM New York, NY, USA, pp. 18–21. DOI: 10.1145/121973.121975.
- [26] Nasser Jazdi. “Cyber physical systems in the context of Industry 4.0”. In: *2014 IEEE international conference on automation, quality and testing, robotics*. IEEE, 2014, pp. 1–4. DOI: 10.1109/AQTR.2014.6857843.
- [27] Pritish Jetley, Lukasz Wesolowski, Filippo Gioachin, Laxmikant V. Kalé, and Thomas R. Quinn. “Scaling hierarchical N-body simulations on GPU clusters”. In: *SC’10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2010, pp. 1–11. DOI: 10.1109/SC.2010.49.
- [28] Zai Jian Jia, Tomás Bautista, Antonio Núñez, Andy D. Pimentel, and Mark Thompson. “A system-level infrastructure for multidimensional MP-SoC design space co-exploration”. In: *ACM Transactions on Embedded Computing Systems* 13.1 (Nov. 2013), pp. 1–26. ISSN: 1539-9087, 1558-3465. DOI: 10.1145/2536747.2536749. URL: <https://dl.acm.org/doi/10.1145/2536747.2536749> (visited on 02/04/2021).

## BIBLIOGRAPHY

---

- [29] Zai Jian Jia, Andy D. Pimentel, Mark Thompson, Tomás Bautista, and Antonio Núñez. “NASA: A generic infrastructure for system-level MP-SoC design space exploration”. In: *2010 8th IEEE Workshop on Embedded Systems for Real-Time Multimedia*. ISSN: 2325-1301 tex.eventtitle: 2010 8th IEEE Workshop on Embedded Systems for Real-Time Multimedia. Oct. 2010, pp. 41–50. DOI: 10.1109/ESTMED.2010.5666979.
- [30] Herman Kelder. *Exploring Scalability in System-Level Simulation Environments for Distributed Cyber-Physical Systems*. Jan. 11, 2023. DOI: 10.13140/RG.2.2.36399.71846.
- [31] Minyoung Kim, Sudarshan Banerjee, Nikil Dutt, and Nalini Venkatasubramanian. “Design space exploration of real-time multi-media MPSoCs with heterogeneous scheduling policies”. In: *Proceedings of the 4th international conference on Hardware/software codesign and system synthesis*. 2006, pp. 16–21. DOI: 10.1145/1176254.1176261.
- [32] Ajay D. Kshemkalyani and Mukesh Singhal. *Distributed computing: principles, algorithms, and systems*. Cambridge University Press, 2011.
- [33] Leslie Lamport and Nancy Lynch. “Distributed computing: Models and methods”. In: *Formal models and semantics*. Elsevier, 1990, pp. 1157–1199.
- [34] Darren R. Law. “Scalable means more than more: a unifying definition of simulation scalability”. In: *1998 Winter Simulation Conference. Proceedings (Cat. No. 98CH36274)*. Vol. 1. IEEE, 1998, pp. 781–788. DOI: 10.1109/WSC.1998.745064.
- [35] Edward A. Lee. “Computing foundations and practice for cyber-physical systems: A preliminary report”. In: *University of California, Berkeley, Tech. Rep. UCB/EECS-2007-72* 21 (2007).
- [36] Edward A. Lee. “The past, present and future of cyber-physical systems: A focus on models”. In: *Sensors* 15.3 (2015). Publisher: MDPI, pp. 4837–4869. DOI: 10.3390/s150304837.
- [37] Yang Liu, Yu Peng, Bailing Wang, Sirui Yao, and Zihe Liu. “Review on cyber-physical systems”. In: *IEEE/CAA Journal of Automatica Sinica* 4.1 (2017). Publisher: IEEE, pp. 27–40. DOI: 10.1109/JAS.2017.7510349.

- [38] Jason Lowe-Power, Abdul Mutaal Ahmad, Ayaz Akram, Mohammad Alian, Rico Amslinger, Matteo Andreozzi, Adrià Armejach, Nils Asmussen, Brad Beckmann, and Srikant Bharadwaj. “The gem5 simulator: Version 20.0+”. In: *arXiv preprint arXiv:2007.03152* (2020). DOI: 10.48550/arxiv.2007.03152.
- [39] *Luigi batch jobs pipeline*. URL: <https://luigi.readthedocs.io/en/stable/> (visited on 02/10/2023).
- [40] Edward A. Luke. “Defining and measuring scalability”. In: *Proceedings of Scalable Parallel Libraries Conference*. IEEE, 1993, pp. 183–186.
- [41] Giovanni Mariani, Gianluca Palermo, Vittorio Zaccaria, and Cristina Silvano. “DeSpErate: Speeding-up design space exploration by using predictive simulation scheduling”. In: *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2014, pp. 1–4.
- [42] Giovanni Mariani, Gianluca Palermo, Vittorio Zaccaria, and Cristina Silvano. “DeSpErate++: An enhanced design space exploration framework using predictive simulation scheduling”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34.2 (2014). Publisher: IEEE, pp. 293–306. DOI: 10.1109/TCAD.2014.2379634.
- [43] Brit Meier, Mladen Skelin, Frans Beenker, and Wouter Leibbrandt. *HTSM Systems Engineering Roadmap*. July 24, 2020.
- [44] Alian Mohammad, Umur Darbaz, Gabor Dozsa, Stephan Diestelhorst, Daehoon Kim, and Nam Sung Kim. “dist-gem5: Distributed simulation of computer clusters”. In: *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2017, pp. 153–162. DOI: 10.1109/ISPASS.2017.7975287.
- [45] *OMNeT++*. original-date: 2018-12-03T16:18:31Z. June 19, 2023. URL: <https://github.com/omnetpp/omnetpp> (visited on 06/20/2023).
- [46] *OMNeT++ simulation distribution*. URL: <https://simdistribution.sourceforge.net/index.html> (visited on 06/29/2023).
- [47] Andy D. Pimentel. “Exploring exploration: A tutorial introduction to embedded systems design space exploration”. In: *IEEE Design & Test* 34.1 (2016). Publisher: IEEE, pp. 77–90. DOI: 10.1109/MDAT.2016.2626445.

## BIBLIOGRAPHY

---

- [48] Andy D. Pimentel, Cagkan Erbas, and Simon Polstra. “A systematic approach to exploring embedded system architectures at multiple abstraction levels”. In: *IEEE transactions on computers* 55.2 (2006). Publisher: IEEE, pp. 99–112. DOI: 10.1109/TC.2006.16.
- [49] Alejandro Rico, Felipe Cabarcas, Carlos Villavieja, Milan Pavlovic, Augusto Vega, Yoav Etsion, Alex Ramirez, and Mateo Valero. “On the simulation of large-scale architectures using multiple application abstraction levels”. In: *ACM Transactions on Architecture and Code Optimization (TACO)* 8.4 (2012). Publisher: ACM New York, NY, USA, pp. 1–20. DOI: 10.1145/2086696.2086715.
- [50] Alejandro Rico, Alejandro Duran, Felipe Cabarcas, Yoav Etsion, Alex Ramirez, and Mateo Valero. “Trace-driven simulation of multithreaded applications”. In: *(IEEE ISPASS) IEEE International Symposium on Performance Analysis of Systems and Software*. IEEE, 2011, pp. 87–96.
- [51] Paul Rosenfeld, Elliott Cooper-Balis, and Bruce Jacob. “DRAMSim2: A cycle accurate memory system simulator”. In: *IEEE computer architecture letters* 10.1 (2011). Publisher: IEEE, pp. 16–19. DOI: 10.1109/L-CA.2011.4.
- [52] Bram van der Sanden, Yonghui Li, Joris van den Aker, Benny Akesson, Tjerk Bijlsma, Martijn Hendriks, Kostas Triantafyllidis, Jacques Verriet, Jeroen Voeten, and Twan Basten. “Model-Driven System-Performance Engineering for Cyber-Physical Systems: Industry Session Paper”. In: *2021 International Conference on Embedded Software (EMSOFT)*. IEEE, 2021, pp. 11–22. DOI: 10.1145/3477244.3477985.
- [53] G. S. Sangeetha, Vignesh Radhakrishnan, Prabhu Prasad, Khyamling Parane, and Basavaraj Talawar. “Trace-driven simulation and design space exploration of network-on-chip topologies on FPGA”. In: *2018 8th International Symposium on Embedded Computing and System Design (ISED)*. IEEE, 2018, pp. 129–134. DOI: 10.1109/ISED.2018.8703884.
- [54] Jianhua Shi, Jiafu Wan, Hehua Yan, and Hui Suo. “A survey of cyber-physical systems”. In: *2011 international conference on wireless communications and signal processing (WCSP)*. IEEE, 2011, pp. 1–6. DOI: 10.1109/WCSP.2011.6096958.
- [55] Rene Steijl and George Barakos. “Sliding mesh algorithm for CFD analysis of helicopter rotor–fuselage aerodynamics”. In: *International journal for numerical methods in fluids* 58.5 (2008). Publisher: Wiley Online Library, pp. 527–549. DOI: 10.1002/flid.1757.

## BIBLIOGRAPHY

---

- [56] Mirko Stoffers, Ralf Bettermann, James Gross, and Klaus Wehrle. *Enabling Distributed Simulation of OMNeT++ INET Models*. Sept. 3, 2014. arXiv: 1409.0994[cs]. URL: <http://arxiv.org/abs/1409.0994> (visited on 06/20/2023).
- [57] Prerit Terway, Kenza Hamidouche, and Niraj K Jha. “DISPATCH: Design Space Exploration of Cyber-Physical Systems”. In: (2020), p. 14.
- [58] UC Davis. *Simulation Research and gem5*. UC Davis Computer Architecture. URL: <https://arch.cs.ucdavis.edu/projects/gem5> (visited on 11/23/2022).
- [59] Andras Varga. *OMNeT++ discrete event simulation system version 6.x user manual*. 2022. URL: <https://doc.omnetpp.org/omnetpp/SimulationManual.pdf> (visited on 11/13/2022).
- [60] David W. Walker and Jack J. Dongarra. “MPI: a standard message passing interface”. In: *Supercomputer 12* (1996). Publisher: ASFRA BV, pp. 56–68.
- [61] Guangxi Wan and Peng Zeng. “Codesign of Architecture, Control, and Scheduling of Modular Cyber-Physical Production Systems for Design Space Exploration”. In: *IEEE Transactions on Industrial Informatics* 18.4 (2021). Publisher: IEEE, pp. 2287–2296. DOI: 10.1109/TII.2021.3097761.
- [62] *Worker — Dask.distributed 2023.6.0 documentation*. URL: <https://distributed.dask.org/en/stable/worker.html> (visited on 06/22/2023).
- [63] Andy B. Yoo, Morris A. Jette, and Mark Grondona. “SLURM: Simple Linux Utility for Resource Management”. In: *Job Scheduling Strategies for Parallel Processing*. Ed. by Dror Feitelson, Larry Rudolph, and Uwe Schwiegelshohn. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2003, pp. 44–60. ISBN: 978-3-540-39727-4. DOI: 10.1007/10968987\_3.





# Appendix A

## Data

The Data appendix contains all raw data collected and utilised during the experiments and visualisations presented in Chapter 5.

### A.1 Experiment: Simulation Campaign

All raw data belonging to the simulation campaign experiment is presented in this section.

#### A.1.1 Baselines

**Table A.1:** Runtime data of sequential baselines with the INET-LANS model.

Timeframe Measure (s) Baseline	Simulation		Compilation	
	Mean	Std	Mean	Std
Sequential	125.09	1.12	0.27	0.01
Sequential Multi	138.09	2.50	0.24	0.07

#### A.1.2 Campaign

**Table A.2:** Runtime data of simulation campaigns with the INET-LANS model. All configurations were set to a single worker process per compute node.

Nodes	Timeframe Measure (s) Evals	Environment		Simulation		Compilation		Worker	
		Mean	Std	Mean	Std	Mean	Std	Mean	Std
1	32	152.41	2.36	142.64	2.23	0.56	0.04	143.22	2.22

Continued on next page

## A. DATA

**Table A.2:** Runtime data of simulation campaigns with the INET-LANS model. All configurations were set to a single worker process per compute node.

Nodes	Timeframe Measure (s) Evals	Environment		Simulation		Compilation		Worker	
		Mean	Std	Mean	Std	Mean	Std	Mean	Std
2	64	292.66	1.96	141.14	2.98	0.41	0.16	141.55	3.10
	96	430.14	1.30	139.54	2.84	0.35	0.18	139.90	2.97
	128	570.51	2.81	139.40	2.52	0.32	0.15	139.71	2.63
	160	713.36	4.21	139.43	2.67	0.29	0.14	139.72	2.76
	192	854.39	3.38	139.29	2.51	0.28	0.14	139.57	2.59
	224	992.79	2.45	139.34	2.30	0.28	0.14	139.62	2.38
	256	1128.89	4.86	138.67	2.51	0.31	0.34	138.99	2.67
	32	153.04	1.92	143.21	2.42	0.55	0.04	143.77	2.42
	64	154.79	4.22	143.18	2.47	0.55	0.08	143.74	2.45
	96	287.82	2.29	140.09	4.63	0.46	0.15	140.56	4.75
	128	297.25	3.34	141.56	3.01	0.42	0.16	141.98	3.11
	160	426.41	3.95	139.77	4.70	0.38	0.17	140.15	4.81
	192	438.80	2.65	140.67	2.87	0.35	0.15	141.02	2.96
	224	567.85	8.49	140.19	4.58	0.56	0.76	140.75	4.88
	256	576.44	1.69	139.92	2.91	0.35	0.22	140.28	3.04
3	32	153.18	1.53	142.90	2.67	0.53	0.04	143.44	2.68
	64	152.48	1.64	139.81	4.77	0.49	0.09	140.31	4.83
	96	158.49	4.41	143.62	2.97	0.55	0.06	144.19	2.96
	128	288.00	2.92	140.51	5.43	0.50	0.14	141.02	5.55
	160	300.24	6.27	140.93	4.32	0.41	0.13	141.35	4.41
	192	300.07	5.92	141.79	3.64	0.36	0.12	142.15	3.72
	224	424.96	6.51	139.63	4.86	0.36	0.14	140.00	4.94
	256	435.65	4.67	140.26	3.80	0.39	0.18	140.65	3.91
4	32	150.64	2.37	142.54	1.99	0.58	0.05	143.13	1.99
	64	152.70	2.14	139.94	5.03	0.51	0.09	140.46	5.10
	96	155.12	3.84	139.77	3.67	0.49	0.07	140.27	3.70
	128	155.45	2.14	143.14	2.26	0.66	0.21	143.81	2.25
	160	286.47	6.02	140.49	6.23	0.42	0.10	140.91	6.30
	192	290.34	2.86	140.14	5.61	0.37	0.10	140.52	5.69
	224	298.78	6.35	141.37	4.56	0.36	0.10	141.74	4.62
	256	300.23	3.49	141.54	3.18	0.42	0.16	141.96	3.29

## A.2 Experiment: Worker Size

**Table A.3:** Runtime data of SMT-enabled simulation campaigns with the INET-LANS model. All configurations were set to a single worker process per compute node.

Nodes	Timeframe Measure (s) Evals	Environment		Simulation		Compilation		Worker	
		Mean	Std	Mean	Std	Mean	Std	Mean	Std
1	32	152.40	1.67	142.65	2.55	0.56	0.04	143.22	2.55
	64	283.85	6.68	260.53	4.37	1.06	0.15	261.61	4.36
	96	414.32	0.59	221.57	56.47	0.86	0.32	222.46	56.79
	128	545.74	7.14	258.13	6.80	0.75	0.32	258.89	6.98
	160	674.21	2.75	236.48	46.84	0.69	0.32	237.19	46.98
	192	808.59	2.99	258.03	7.51	0.67	0.35	258.71	7.64
	224	930.96	4.95	241.76	39.87	0.64	0.32	242.40	39.96
	256	1066.48	9.32	257.00	7.04	0.62	0.28	257.62	7.15
2	32	141.57	1.35	134.80	1.64	0.42	0.04	135.22	1.64
	64	283.68	5.23	259.81	4.25	1.09	0.13	260.92	4.24
	96	285.07	2.32	221.71	55.19	0.89	0.28	222.62	55.46
	128	290.93	6.46	261.67	4.44	1.31	0.61	263.03	4.46
	160	404.08	1.93	236.56	51.51	0.91	0.27	237.50	51.74
	192	429.66	23.68	224.30	55.25	0.87	0.35	225.20	55.57
	224	498.48	2.88	232.51	40.23	0.79	0.34	233.32	40.52
	256	561.04	8.32	258.96	7.35	0.74	0.33	259.72	7.53
3	32	141.31	1.32	131.61	3.25	0.38	0.05	131.99	3.24
	64	285.01	1.31	260.52	4.54	1.07	0.14	261.63	4.53
	96	283.50	5.29	219.63	58.87	0.87	0.32	220.52	59.17
	128	282.93	4.10	203.88	57.36	0.85	0.28	204.74	57.62
	160	245.58	1.36	211.38	26.25	0.74	0.15	212.15	26.26
	192	293.25	4.65	262.59	4.39	0.92	0.21	263.56	4.38
	224	405.51	3.32	243.62	45.98	1.13	0.60	244.79	46.26
	256	411.29	2.40	231.68	54.57	0.82	0.29	232.54	54.81
4	32	135.60	0.65	128.45	1.78	0.35	0.03	128.80	1.79
	64	282.54	3.45	260.62	3.97	1.08	0.14	261.73	3.96
	96	287.57	6.13	218.99	60.59	0.82	0.31	219.84	60.90
	128	284.81	1.86	202.20	60.70	0.76	0.28	202.98	60.97
	160	210.90	2.08	170.17	23.84	0.55	0.08	170.74	23.83
	192	224.77	4.89	190.85	25.32	0.68	0.27	191.56	25.31
	224	251.78	0.58	221.35	26.42	0.80	0.23	222.19	26.42
	256	291.76	2.67	262.10	4.12	0.81	0.25	262.96	4.10

## A.2 Experiment: Worker Size

All raw data belonging to the worker size experiment is presented in this section.

## A. DATA

**Table A.4:** Runtime data of simulation campaigns with the INET-LANS model. All non-single worker runtime data and the corresponding single worker runtime data.

Nodes	Workers	Timeframe	Environment		Simulation		Compilation		Worker	
		Measure (s)	Mean	Std	Mean	Std	Mean	Std	Mean	Std
		Evals								
1	1	32	152.41	2.36	142.64	2.23	0.56	0.04	143.22	2.22
		64	292.66	1.96	141.14	2.98	0.41	0.16	141.55	3.10
		96	430.14	1.30	139.54	2.84	0.35	0.18	139.90	2.97
		128	570.51	2.81	139.40	2.52	0.32	0.15	139.71	2.63
	2	32	153.60	2.48	143.48	2.57	0.45	0.08	143.93	2.58
		64	291.52	1.03	141.11	2.60	0.41	0.25	141.53	2.72
	4	32	152.25	3.12	143.32	2.33	0.44	0.08	143.76	2.32
		64	292.27	1.51	140.75	3.03	0.33	0.10	141.08	3.09
	8	32	154.05	4.35	144.18	2.42	0.43	0.07	144.61	2.42
		64	296.26	2.33	142.29	3.31	0.34	0.11	142.63	3.38
	16	32	153.40	1.39	145.03	2.14	0.45	0.05	145.48	2.15
		64	296.57	2.14	143.01	2.95	0.35	0.11	143.36	3.01
	32	32	158.06	5.46	147.91	2.91	0.50	0.05	148.42	2.92
		64	304.42	2.74	145.63	3.28	0.37	0.12	146.00	3.35
2	1	32	153.04	1.92	143.21	2.42	0.55	0.04	143.77	2.42
		64	154.79	4.22	143.18	2.47	0.55	0.08	143.74	2.45
		96	287.82	2.29	140.09	4.63	0.46	0.15	140.56	4.75
		128	297.25	3.34	141.56	3.01	0.42	0.16	141.98	3.11
	2	32	148.42	3.97	138.11	4.79	0.42	0.07	138.54	4.83
		64	157.51	2.58	143.70	2.68	0.47	0.06	144.18	2.68
	4	32	151.63	2.68	140.03	6.18	0.40	0.07	140.43	6.21
		64	154.23	1.07	144.15	2.54	0.44	0.07	144.58	2.54
	8	32	146.58	3.14	138.97	4.47	0.39	0.07	139.36	4.49
		64	154.06	2.01	143.83	2.36	0.48	0.06	144.31	2.36
	16	32	153.65	5.01	142.94	5.29	0.42	0.06	143.36	5.32
		64	156.88	2.45	145.31	2.38	0.51	0.04	145.81	2.38
	32	32	154.91	6.73	144.32	6.31	0.45	0.05	144.78	6.34
		64	158.81	1.59	147.95	2.36	0.47	0.07	148.42	2.37
3	1	32	153.18	1.53	142.90	2.67	0.53	0.04	143.44	2.68
		64	152.48	1.64	139.81	4.77	0.49	0.09	140.31	4.83
		96	158.49	4.41	143.62	2.97	0.55	0.06	144.19	2.96
		128	288.00	2.92	140.51	5.43	0.50	0.14	141.02	5.55
	2	32	147.48	4.32	136.04	6.05	0.40	0.07	136.45	6.08
		64	150.34	1.86	138.54	3.93	0.43	0.06	138.97	3.95
		96	156.78	3.25	143.78	2.79	0.41	0.06	144.19	2.79
	4	32	151.18	12.61	137.35	6.25	0.40	0.06	137.74	6.27
		64	153.19	2.40	141.22	5.15	0.46	0.08	141.68	5.16

Continued on next page

### A.3 Experiment: PDES

**Table A.4:** Runtime data of simulation campaigns with the INET-LANS model. All non-single worker runtime data and the corresponding single worker runtime data.

Nodes	Workers	Timeframe Measure (s) Evals	Environment		Simulation		Compilation		Worker	
			Mean	Std	Mean	Std	Mean	Std	Mean	Std
4	8	96	156.30	2.34	143.91	2.52	0.38	0.06	144.29	2.51
		32	150.09	5.09	140.26	5.86	0.41	0.07	140.67	5.90
		64	155.01	2.03	142.53	4.86	0.45	0.05	142.98	4.88
	16	96	157.01	0.82	144.69	2.46	0.71	0.70	145.40	2.50
		32	150.55	5.48	140.92	5.55	0.44	0.07	141.36	5.61
		64	156.47	3.68	144.82	4.53	0.47	0.06	145.29	4.54
	32	96	157.07	1.28	145.34	2.23	0.38	0.06	145.72	2.23
		32	156.18	5.12	142.80	5.85	0.39	0.06	143.19	5.87
		64	160.67	3.50	145.51	5.04	0.44	0.08	145.95	5.08
	1	96	162.63	3.08	148.30	2.69	0.39	0.07	148.69	2.68
		32	150.64	2.37	142.54	1.99	0.58	0.05	143.13	1.99
		64	152.70	2.14	139.94	5.03	0.51	0.09	140.46	5.10
	2	96	155.12	3.84	139.77	3.67	0.49	0.07	140.27	3.70
		128	155.45	2.14	143.14	2.26	0.66	0.21	143.81	2.25
		32	151.74	13.41	136.29	6.58	0.41	0.07	136.71	6.62
	4	64	155.79	9.12	137.39	5.20	0.45	0.08	137.84	5.24
		128	156.95	2.40	143.89	2.42	0.38	0.08	144.28	2.41
		32	150.03	4.50	136.03	5.75	0.37	0.07	136.40	5.77
	8	64	151.26	2.97	139.44	5.14	0.41	0.05	139.85	5.16
		128	159.48	3.57	144.21	2.87	0.36	0.05	144.57	2.88
		32	148.01	6.11	139.20	5.66	0.42	0.08	139.62	5.70
	16	64	153.00	5.42	140.01	5.01	0.44	0.07	140.44	5.04
		128	160.25	4.21	144.53	2.56	0.41	0.13	144.94	2.55
		32	153.60	3.47	142.82	5.53	0.44	0.06	143.26	5.56
	32	64	156.30	3.65	141.16	6.19	0.47	0.09	141.63	6.22
		128	157.41	1.15	145.80	2.43	0.36	0.05	146.16	2.44
		32	157.86	3.08	147.22	3.22	0.44	0.07	147.65	3.23
		64	170.71	14.72	145.29	7.21	0.43	0.08	145.73	7.24
		128	169.83	13.74	148.64	3.24	0.37	0.05	149.00	3.24

## A.3 Experiment: PDES

All raw data belonging to the PDES experiment is presented in this section.

### A.3.1 Baselines

## A. DATA

**Table A.5:** Runtime data of baselines with the communication intensive CQN model.

Timeframe Measure (s) Baseline	Simulation		Compilation	
	Mean	Std	Mean	Std
Sequential	40.86	0.52	4.41	0.23
Sequential Multi	57.97	4.91	4.26	0.60
Sequential PDES	8.56	0.15	4.34	0.31
Sequential PDES Multi	9.96	0.52	4.53	0.32

**Table A.6:** Runtime data of baselines with the computation intensive CQN model.

Timeframe Measure (s) Baseline	Simulation		Compilation	
	Mean	Std	Mean	Std
Sequential	26.72	0.29	4.35	0.38
Sequential Multi	36.99	2.31	4.39	0.50
Sequential PDES	6.70	0.23	4.32	0.31
Sequential PDES Multi	7.53	0.35	4.59	0.34

**Table A.7:** Runtime data of baselines with the subsystem intensive CQN model.

Timeframe Measure (s) Baseline	Simulation		Compilation	
	Mean	Std	Mean	Std
Sequential	41.78	0.43	4.68	0.51
Sequential Multi	59.85	4.86	4.26	0.60
Sequential PDES	8.50	0.17	4.41	0.32
Sequential PDES Multi	9.74	0.54	4.62	0.30

### A.3.2 Campaign: Sequential

**Table A.8:** Runtime data of simulation campaigns with the sequential communication intensive CQN model. All configurations were set to a single worker process per compute node.

Nodes	Timeframe Measure (s) Evals	Environment		Simulation		Compilation		Worker	
		Mean	Std	Mean	Std	Mean	Std	Mean	Std
1	4	53.67	3.73	43.22	2.82	4.91	0.18	48.13	2.80

Continued on next page

### A.3 Experiment: PDES

**Table A.8:** Runtime data of simulation campaigns with the sequential communication intensive CQN model. All configurations were set to a single worker process per compute node.

Nodes	Timeframe Measure (s) Evals	Environment		Simulation		Compilation		Worker	
		Mean	Std	Mean	Std	Mean	Std	Mean	Std
2	8	51.11	1.47	41.88	1.12	4.95	0.31	46.83	1.11
	12	60.41	3.17	47.34	4.66	4.96	0.33	52.30	4.67
	16	60.72	1.74	50.58	1.28	5.01	0.37	55.58	1.31
	20	66.98	3.26	52.61	2.97	5.01	0.37	57.62	3.00
	24	64.87	0.83	54.44	0.94	5.00	0.41	59.45	0.99
	28	67.23	2.43	53.89	2.00	4.91	0.31	58.80	2.01
	32	66.57	0.95	55.97	1.00	5.03	0.22	61.00	0.98
	64	130.50	3.70	56.07	1.66	4.16	0.87	60.23	1.82
	96	188.56	3.62	55.93	1.54	3.82	0.78	59.75	1.76
	128	252.56	5.87	56.15	1.86	3.76	0.82	59.91	2.03
	256	497.18	5.60	56.35	2.34	3.49	0.62	59.84	2.42
	4	53.97	2.92	43.46	2.42	4.83	0.24	48.29	2.50
	8	53.23	3.42	42.21	2.10	4.89	0.15	47.10	2.09
	12	56.22	4.52	42.87	2.86	4.87	0.28	47.74	2.86
	16	51.98	1.73	42.04	1.21	4.91	0.24	46.96	1.22
	20	60.66	2.08	45.07	4.45	4.92	0.32	49.99	4.50
	24	62.25	3.22	47.54	4.52	5.03	0.31	52.57	4.52
	28	61.92	3.23	49.03	3.24	5.00	0.37	54.03	3.25
	32	67.27	1.91	55.91	1.14	5.17	0.25	61.08	1.13
	64	71.81	4.31	56.15	1.61	5.11	0.26	61.26	1.63
3	96	124.07	4.17	54.34	3.06	4.51	0.81	58.85	3.74
	128	134.67	0.78	56.32	1.81	4.24	0.90	60.57	1.95
	256	257.78	4.52	56.48	2.41	3.77	0.83	60.25	2.52
	4	53.54	3.62	42.90	2.83	4.64	0.34	47.54	3.01
	8	53.70	2.45	43.12	2.27	4.83	0.26	47.95	2.32
	12	55.17	3.69	42.92	2.46	4.90	0.23	47.83	2.51
	16	58.78	2.21	43.11	2.90	4.88	0.29	47.99	2.90
	20	58.18	4.08	42.30	2.40	4.91	0.29	47.21	2.43
	24	53.56	1.71	42.31	1.47	4.93	0.26	47.24	1.46
	28	63.60	0.87	44.74	4.66	4.91	0.32	49.65	4.72
	32	67.74	3.86	55.95	1.47	5.04	0.30	60.98	1.52
	64	67.13	1.28	53.18	3.13	5.03	0.33	58.21	3.19
	96	71.36	3.34	56.20	1.50	4.99	0.25	61.19	1.52
	128	126.04	5.10	54.26	4.45	4.79	0.79	59.05	4.99
	256	193.77	3.05	55.77	2.47	4.02	0.87	59.79	2.75
4	4	51.69	2.69	42.27	1.89	4.58	0.30	46.85	1.83
	8	55.70	1.78	44.35	2.81	4.83	0.28	49.19	2.82
	12	56.10	0.82	43.87	2.81	4.80	0.27	48.67	2.76

Continued on next page

## A. DATA

**Table A.8:** Runtime data of simulation campaigns with the sequential communication intensive CQN model. All configurations were set to a single worker process per compute node.

Nodes	Timeframe	Environment		Simulation		Compilation		Worker	
	Measure (s) Evals	Mean	Std	Mean	Std	Mean	Std	Mean	Std
	16	53.94	2.27	42.50	1.92	4.91	0.21	47.41	1.92
	20	56.34	3.14	42.84	2.43	4.91	0.24	47.75	2.42
	24	55.96	5.11	42.43	2.19	5.13	0.69	47.57	2.22
	28	60.63	3.90	43.95	3.84	4.92	0.24	48.87	3.87
	32	68.65	3.55	55.96	1.42	5.11	0.27	61.07	1.43
	64	67.40	1.85	52.33	5.01	5.05	0.29	57.38	5.04
	96	69.47	1.43	54.75	1.26	4.94	0.39	59.69	1.29
	128	71.44	4.54	56.27	1.45	5.17	0.25	61.44	1.46
	256	134.94	3.82	56.35	1.50	4.43	1.18	60.78	2.03

**Table A.9:** Runtime data of simulation campaigns with the sequential computation intensive CQN model. All configurations were set to a single worker process per compute node.

Nodes	Timeframe	Environment		Simulation		Compilation		Worker	
	Measure (s) Evals	Mean	Std	Mean	Std	Mean	Std	Mean	Std
1	4	34.96	0.66	27.03	0.59	4.87	0.17	31.90	0.59
	8	35.49	0.30	27.14	0.50	4.82	0.34	31.97	0.64
	12	43.16	0.65	30.79	3.03	5.01	0.36	35.79	3.08
	16	43.56	0.69	34.18	1.26	5.00	0.38	39.18	1.22
	20	46.83	0.75	36.36	1.95	5.00	0.37	41.36	1.97
	24	48.36	0.91	38.24	1.14	4.92	0.38	43.17	1.15
	28	47.16	2.47	37.34	2.28	4.90	0.28	42.24	2.30
	32	45.77	0.69	35.82	0.72	5.09	0.23	40.91	0.71
	64	86.22	2.05	35.97	0.65	4.25	0.91	40.22	1.04
	96	125.05	1.04	35.94	0.63	3.86	0.81	39.80	0.96
	128	166.20	0.95	36.01	0.69	3.79	0.79	39.79	1.04
	256	325.63	0.67	36.05	0.65	3.55	0.63	39.60	0.87
2	4	36.14	0.88	27.73	0.87	4.74	0.31	32.47	0.90
	8	36.37	1.19	27.25	0.79	4.87	0.16	32.12	0.83
	12	41.38	3.83	27.69	2.34	4.87	0.28	32.57	2.29
	16	36.90	0.81	27.19	0.84	4.89	0.30	32.07	0.93
	20	42.72	0.91	29.29	2.89	4.95	0.32	34.24	2.91
	24	43.24	0.50	30.92	3.01	4.97	0.36	35.89	3.03
	28	44.12	0.37	32.72	2.70	5.02	0.36	37.74	2.67
	32	45.09	0.50	35.79	0.69	5.02	0.22	40.81	0.71

Continued on next page



### A.3 Experiment: PDES

**Table A.9:** Runtime data of simulation campaigns with the sequential computation intensive CQN model. All configurations were set to a single worker process per compute node.

Nodes	Timeframe Measure (s) Evals	Environment		Simulation		Compilation		Worker	
		Mean	Std	Mean	Std	Mean	Std	Mean	Std
3	64	46.57	0.31	35.90	0.61	5.15	0.27	41.05	0.64
	96	85.19	1.70	35.40	1.37	4.57	0.73	39.97	1.84
	128	87.33	1.03	35.93	0.62	4.33	0.98	40.27	1.12
	256	167.48	0.72	36.10	0.78	3.85	0.81	39.96	1.19
	4	35.48	0.88	27.44	0.81	4.64	0.42	32.08	0.85
	8	36.46	1.16	27.49	1.05	4.78	0.29	32.27	0.99
	12	36.03	0.53	27.26	0.56	4.82	0.26	32.08	0.53
	16	41.03	3.83	27.59	1.87	4.81	0.31	32.39	1.93
	20	39.77	2.40	27.37	1.32	4.87	0.31	32.24	1.36
	24	37.23	1.42	27.17	0.76	5.04	0.40	32.22	0.87
	28	41.74	0.69	28.86	2.54	4.88	0.32	33.74	2.56
	32	45.30	0.56	35.79	0.66	5.03	0.24	40.82	0.66
	64	47.74	0.97	35.25	1.83	5.07	0.33	40.32	1.87
	96	46.53	0.57	35.86	0.60	4.92	0.26	40.78	0.61
	128	85.65	1.63	35.03	2.64	4.83	0.70	39.86	3.09
	256	129.13	1.83	36.28	1.23	4.25	0.90	40.54	1.43
4	4	35.02	0.64	26.99	0.60	4.41	0.34	31.40	0.69
	8	36.82	1.47	27.70	1.04	4.79	0.28	32.49	1.07
	12	37.05	0.93	27.41	0.99	4.79	0.26	32.20	1.04
	16	37.23	1.19	27.26	0.89	4.83	0.26	32.09	0.89
	20	38.22	1.00	27.56	1.26	4.82	0.28	32.38	1.19
	24	39.75	2.49	27.35	1.42	4.88	0.27	32.23	1.47
	28	42.89	3.32	27.70	2.01	4.84	0.32	32.54	2.02
	32	45.22	0.67	35.79	0.71	5.02	0.25	40.82	0.70
	64	46.96	1.14	33.68	3.48	5.09	0.32	38.78	3.57
	96	48.89	0.82	38.36	0.72	4.96	0.39	43.32	0.80
	128	48.64	2.15	36.29	0.82	6.61	1.87	42.91	1.98
	256	88.39	0.81	35.99	0.65	4.33	0.87	40.33	1.06

**Table A.10:** Runtime data of simulation campaigns with the sequential subsystem intensive CQN model. All configurations were set to a single worker process per compute node.

Nodes	Timeframe Measure (s) Evals	Environment		Simulation		Compilation		Worker	
		Mean	Std	Mean	Std	Mean	Std	Mean	Std
1	4	52.92	4.05	43.38	2.68	4.88	0.24	48.26	2.67

Continued on next page

## A. DATA

**Table A.10:** Runtime data of simulation campaigns with the sequential subsystem intensive CQN model. All configurations were set to a single worker process per compute node.

Nodes	Timeframe Measure (s) Evals	Environment		Simulation		Compilation		Worker	
		Mean	Std	Mean	Std	Mean	Std	Mean	Std
2	8	51.71	0.52	43.00	0.68	4.82	0.35	47.82	0.80
	12	60.80	2.97	48.59	4.41	4.99	0.30	53.57	4.44
	16	63.03	2.88	51.79	1.30	5.05	0.37	56.84	1.32
	20	68.82	2.23	53.86	3.07	4.99	0.42	58.85	3.04
	24	66.49	1.94	55.75	0.92	4.97	0.38	60.73	1.03
	28	66.95	2.40	55.09	1.92	4.94	0.30	60.03	1.95
	32	68.76	1.60	57.29	1.23	4.94	0.23	62.23	1.21
	64	132.64	4.98	57.24	1.57	4.20	0.90	61.44	1.84
	96	197.35	3.92	57.38	2.14	3.94	0.90	61.33	2.27
	128	256.81	5.11	57.27	1.82	3.75	0.79	61.02	1.96
	256	512.74	7.31	57.56	2.64	3.66	1.08	61.23	2.91
	4	54.25	0.95	44.49	2.06	4.93	0.11	49.42	2.10
	8	55.18	1.31	43.90	2.05	4.89	0.15	48.79	2.00
	12	59.37	4.00	44.11	3.45	4.87	0.31	48.98	3.45
	16	54.00	2.10	43.51	1.31	4.93	0.28	48.44	1.33
	20	62.67	3.02	46.25	4.65	4.99	0.25	51.24	4.69
	24	62.52	2.24	48.50	4.52	5.02	0.31	53.52	4.52
	28	63.01	2.08	50.24	3.42	4.95	0.37	55.19	3.47
	32	67.87	3.47	57.21	1.49	5.01	0.22	62.21	1.47
	64	70.81	3.71	57.38	1.42	5.11	0.28	62.49	1.42
3	96	127.28	3.54	55.61	3.08	4.54	0.81	60.15	3.74
	128	137.08	4.16	57.33	1.75	4.53	1.28	61.86	2.21
	256	265.68	2.28	57.55	2.11	4.30	1.44	61.85	2.68
	4	57.07	2.04	45.02	3.15	4.64	0.34	49.67	3.38
	8	56.63	2.70	44.53	2.94	4.86	0.22	49.39	2.94
	12	54.26	2.25	43.39	1.75	4.87	0.27	48.26	1.72
	16	56.59	3.21	43.49	2.37	4.87	0.29	48.36	2.41
	20	57.98	3.71	44.07	2.96	4.83	0.32	48.90	3.00
	24	54.10	1.18	43.34	1.15	4.97	0.28	48.31	1.18
	28	64.16	2.30	45.87	4.46	5.27	0.75	51.14	4.59
	32	70.73	3.85	57.21	1.65	4.96	0.24	62.17	1.67
	64	69.69	3.09	55.07	3.19	5.02	0.31	60.09	3.22
	96	69.84	2.91	57.41	1.20	4.93	0.26	62.34	1.21
	128	131.22	3.92	55.73	4.41	4.74	0.78	60.47	4.88
	256	198.20	1.38	57.14	2.67	4.11	0.92	61.24	2.86
4	4	52.89	3.93	43.00	2.40	4.38	0.38	47.38	2.48
	8	56.42	1.37	45.44	2.25	4.87	0.20	50.32	2.30
	12	55.11	3.36	43.60	2.32	4.75	0.30	48.36	2.38

Continued on next page

### A.3 Experiment: PDES

**Table A.10:** Runtime data of simulation campaigns with the sequential subsystem intensive CQN model. All configurations were set to a single worker process per compute node.

Nodes	Timeframe	Environment		Simulation		Compilation		Worker	
	Measure (s)	Mean	Std	Mean	Std	Mean	Std	Mean	Std
	Evals								
	16	56.63	2.64	44.04	2.45	4.87	0.26	48.92	2.49
	20	58.91	1.78	44.41	2.92	4.86	0.27	49.27	2.93
	24	57.99	5.67	43.87	2.62	4.91	0.24	48.78	2.61
	28	60.89	1.37	44.95	3.39	4.89	0.31	49.84	3.40
	32	70.57	4.31	57.43	1.87	5.17	0.25	62.60	1.86
	64	70.52	4.70	53.31	5.38	5.48	0.82	58.80	5.52
	96	71.76	2.65	56.18	1.30	4.89	0.38	61.08	1.33
	128	71.29	1.29	57.35	0.96	5.77	1.27	63.12	1.56
	256	138.77	3.92	57.51	1.42	4.42	0.82	61.93	1.68

#### A.3.3 Campaign: PDES

**Table A.11:** Runtime data of simulation campaigns with the PDES-enabled communication intensive CQN model. All configurations were set to a single worker process per compute node.

Nodes	Timeframe	Environment		Simulation		Compilation		Worker	
	Measure (s)	Mean	Std	Mean	Std	Mean	Std	Mean	Std
	Evals								
1	4	18.43	0.71	10.29	0.59	4.94	0.18	15.23	0.66
	8	33.22	1.11	10.22	0.62	4.70	0.32	14.92	0.76
	12	47.66	0.98	10.11	0.71	4.62	0.35	14.73	0.81
	16	71.14	7.58	12.22	2.18	4.52	0.41	16.74	2.11
	20	93.90	8.06	13.40	2.02	4.59	0.35	17.98	2.02
	24	108.65	11.19	12.89	2.23	4.58	0.37	17.47	2.28
	28	108.59	4.06	10.42	0.89	4.55	0.34	14.97	0.94
	32	140.55	16.66	12.44	2.26	4.58	0.34	17.02	2.24
2	4	17.63	0.79	9.43	0.52	4.90	0.30	14.33	0.67
	8	18.36	0.61	10.02	0.54	4.88	0.21	14.90	0.64
	12	36.05	4.43	10.66	1.90	4.72	0.37	15.38	1.87
	16	37.81	4.02	11.12	2.06	4.68	0.37	15.80	2.10
	20	56.90	9.75	11.31	2.83	4.65	0.38	15.96	2.81
	24	47.88	1.14	9.81	0.57	4.62	0.44	14.42	0.76
	28	66.90	10.38	10.53	1.94	4.66	0.43	15.19	2.01
	32	63.04	1.56	9.82	0.63	4.65	0.35	14.47	0.78
3	4	16.99	0.33	9.08	0.30	4.71	0.37	13.79	0.53

Continued on next page

## A. DATA

**Table A.11:** Runtime data of simulation campaigns with the PDES-enabled communication intensive CQN model. All configurations were set to a single worker process per compute node.

Nodes	Timeframe Measure (s) Evals	Environment		Simulation		Compilation		Worker	
		Mean	Std	Mean	Std	Mean	Std	Mean	Std
4	8	17.91	0.22	9.58	0.41	4.88	0.24	14.46	0.54
	12	19.16	1.35	10.38	1.10	4.87	0.23	15.25	1.15
	16	49.36	37.85	10.99	9.34	4.76	0.31	15.75	9.29
	20	36.39	2.86	10.50	1.51	4.77	0.28	15.27	1.50
	24	33.37	0.60	10.05	0.55	4.73	0.34	14.78	0.74
	28	86.78	48.25	11.94	9.96	4.69	0.34	16.63	9.98
	32	54.75	16.88	9.76	0.64	4.65	0.35	14.40	0.78
	4	16.76	0.09	8.90	0.23	4.61	0.27	13.52	0.37
	8	17.84	0.70	9.42	0.49	4.83	0.23	14.25	0.54
	12	22.06	1.21	10.74	2.12	4.85	0.24	15.59	2.07
	16	22.37	1.36	11.02	2.06	4.83	0.30	15.86	2.07
	20	118.27	1.05	14.68	18.08	4.78	0.30	19.46	18.03
	24	32.82	0.39	9.91	0.69	4.79	0.29	14.70	0.85
	28	39.54	2.08	10.70	1.83	4.70	0.35	15.40	1.90
	32	79.23	75.84	10.58	1.48	4.71	0.35	15.29	1.52

**Table A.12:** Runtime data of simulation campaigns with the PDES-enabled computation intensive CQN model. All configurations were set to a single worker process per compute node.

Nodes	Timeframe Measure (s) Evals	Environment		Simulation		Compilation		Worker	
		Mean	Std	Mean	Std	Mean	Std	Mean	Std
1	4	16.10	0.92	8.13	0.58	4.94	0.22	13.07	0.71
	8	27.78	0.27	7.71	0.38	4.62	0.34	12.33	0.45
	12	39.81	1.33	7.49	0.47	4.66	0.31	12.14	0.64
	16	53.04	2.04	7.91	0.65	4.63	0.29	12.54	0.62
	20	84.92	2.85	11.57	1.13	4.67	0.35	16.24	1.20
	24	76.45	0.31	7.61	0.62	4.58	0.35	12.20	0.77
	28	92.63	7.48	8.28	1.18	4.52	0.36	12.80	1.21
	32	125.98	15.91	10.84	2.04	4.53	0.37	15.37	2.05
2	4	15.42	0.41	7.29	0.32	5.08	0.39	12.37	0.52
	8	15.81	0.55	7.84	0.40	4.82	0.32	12.66	0.57
	12	27.71	0.54	7.63	0.56	4.76	0.33	12.38	0.74
	16	29.26	2.49	7.81	0.96	4.62	0.37	12.43	1.10

Continued on next page

### A.3 Experiment: PDES

**Table A.12:** Runtime data of simulation campaigns with the PDES-enabled computation intensive CQN model. All configurations were set to a single worker process per compute node.

Nodes	Timeframe Measure (s) Evals	Environment		Simulation		Compilation		Worker	
		Mean	Std	Mean	Std	Mean	Std	Mean	Std
3	20	46.53	9.71	8.50	2.50	4.66	0.30	13.16	2.52
	24	40.49	0.74	7.60	0.50	4.63	0.35	12.23	0.58
	28	64.24	12.43	8.88	2.73	4.62	0.31	13.50	2.74
	32	56.15	8.36	8.01	1.48	4.56	0.36	12.57	1.48
	4	14.84	0.26	6.94	0.29	4.67	0.33	11.61	0.42
	8	15.74	0.46	7.44	0.40	4.90	0.27	12.33	0.50
	12	16.14	0.26	7.85	0.45	4.88	0.23	12.72	0.56
	16	54.01	36.33	9.70	9.81	4.79	0.30	14.49	9.75
4	20	28.79	1.65	7.66	0.49	4.91	0.67	12.57	0.96
	24	28.71	0.62	7.63	0.58	4.70	0.36	12.34	0.71
	28	41.73	4.57	7.73	0.95	4.71	0.28	12.44	1.02
	32	44.53	6.28	8.07	1.47	4.63	0.35	12.70	1.51
	4	14.37	0.15	6.73	0.22	4.50	0.24	11.23	0.27
	8	15.32	0.30	7.16	0.36	4.80	0.28	11.96	0.38
	12	15.97	0.17	7.61	0.43	4.87	0.21	12.49	0.48
	16	20.23	1.02	8.71	2.12	4.84	0.32	13.55	2.20
	20	72.55	0.70	12.87	12.34	4.94	0.43	17.81	12.18
	24	27.98	0.18	7.56	0.49	4.75	0.31	12.31	0.65
	28	35.46	0.13	8.52	1.89	4.71	0.41	13.23	1.97
	32	30.96	1.62	7.97	0.98	4.69	0.37	12.65	1.07

**Table A.13:** Runtime data of simulation campaigns with the PDES-enabled subsystem intensive CQN model. All configurations were set to a single worker process per compute node.

Nodes	Timeframe Measure (s) Evals	Environment		Simulation		Compilation		Worker	
		Mean	Std	Mean	Std	Mean	Std	Mean	Std
1	4	17.35	0.58	9.64	0.52	4.75	0.35	14.39	0.65
	8	31.51	0.79	9.59	0.45	4.62	0.36	14.21	0.59
	12	46.65	1.14	9.67	0.52	4.69	0.28	14.36	0.63
	16	63.69	6.50	10.41	1.48	4.62	0.35	15.03	1.55
	20	75.60	2.28	9.70	0.61	4.68	0.33	14.38	0.71
	24	89.54	1.71	9.67	0.66	4.58	0.37	14.25	0.78
	28	108.19	5.14	10.35	1.03	4.59	0.34	14.94	1.06
	32	126.16	15.35	10.57	1.89	4.62	0.30	15.19	1.90

Continued on next page

## A. DATA

**Table A.13:** Runtime data of simulation campaigns with the PDES-enabled subsystem intensive CQN model. All configurations were set to a single worker process per compute node.

Nodes	Timeframe Measure (s) Evals	Environment		Simulation		Compilation		Worker	
		Mean	Std	Mean	Std	Mean	Std	Mean	Std
2	4	19.59	3.05	10.29	2.49	4.85	0.25	15.14	2.51
	8	18.20	0.41	9.80	0.51	4.93	0.28	14.74	0.66
	12	31.64	0.82	9.47	0.58	4.71	0.36	14.18	0.76
	16	32.99	0.89	9.73	0.68	4.69	0.34	14.42	0.83
	20	54.84	9.27	10.92	2.58	4.68	0.35	15.60	2.60
	24	47.06	0.89	9.68	0.69	4.64	0.34	14.32	0.82
	28	63.32	8.05	9.81	1.56	4.59	0.33	14.40	1.61
	32	60.90	2.80	9.54	0.66	4.53	0.39	14.07	0.82
3	4	16.71	0.41	8.75	0.27	4.67	0.33	13.42	0.51
	8	17.55	0.37	9.27	0.52	4.77	0.31	14.05	0.62
	12	18.42	0.46	9.70	0.53	4.90	0.23	14.60	0.60
	16	30.96	0.52	9.44	0.68	4.79	0.32	14.23	0.89
	20	34.81	2.75	10.02	1.27	4.72	0.35	14.74	1.30
	24	33.14	1.38	9.61	0.54	4.83	0.50	14.44	0.77
	28	81.79	50.58	11.12	9.96	4.63	0.37	15.75	9.98
	32	47.26	1.97	9.55	0.65	4.69	0.31	14.24	0.77
4	4	16.22	0.20	8.62	0.14	4.41	0.42	13.03	0.43
	8	17.41	0.45	9.13	0.39	4.92	0.16	14.05	0.42
	12	18.05	0.40	9.41	0.58	4.83	0.23	14.24	0.68
	16	18.62	0.86	9.80	0.60	4.84	0.29	14.64	0.71
	20	75.06	0.32	12.09	9.02	4.81	0.30	16.90	8.98
	24	32.38	0.32	9.74	0.66	4.80	0.30	14.54	0.85
	28	39.74	1.78	10.51	1.99	4.90	0.59	15.42	1.95
	32	36.70	1.29	10.20	1.21	4.73	0.39	14.93	1.31

## Appendix B

# Visualisations

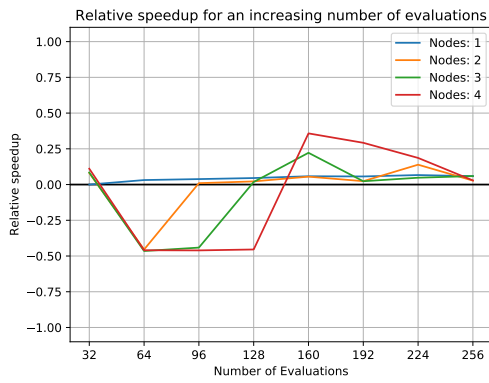
The Visualisation appendix contains additional informational figures generated during the experiments, but not presented (or presented differently) in Chapter 5.

## B. VISUALISATIONS

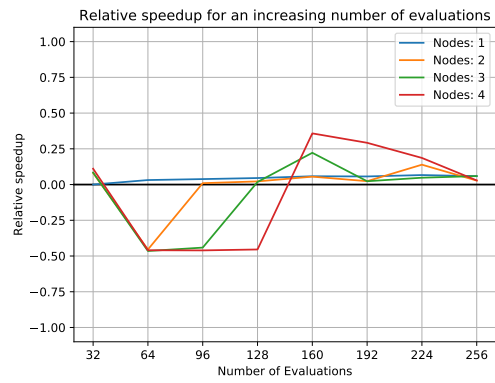
### B.1 Experiment: Simulation Campaign

All additional figures belonging to the simulation campaign experiment are presented in this section.

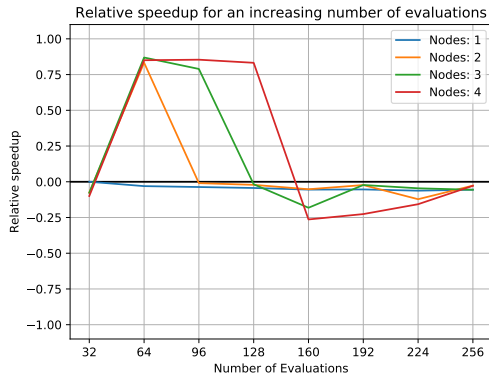
#### B.1.1 SMT Enabled



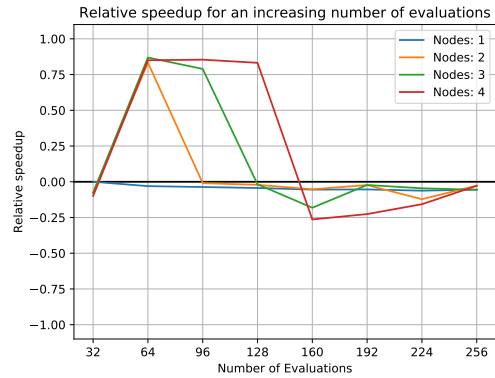
(a) Relative speedup for SMT against non-SMT with *Sequential* baseline and strong scaling ( $S_{rel}(SMT, regular)$ )



(b) Relative speedup for SMT against non-SMT with *Sequential Multi* baseline and strong scaling ( $S_{rel}(SMT, regular)$ )



(c) Relative speedup for non-SMT against SMT with *Sequential* baseline and strong scaling ( $S_{rel}(regular, SMT)$ )



(d) Relative speedup for non-SMT against SMT with *Sequential Multi* baseline and strong scaling ( $S_{rel}(regular, SMT)$ )

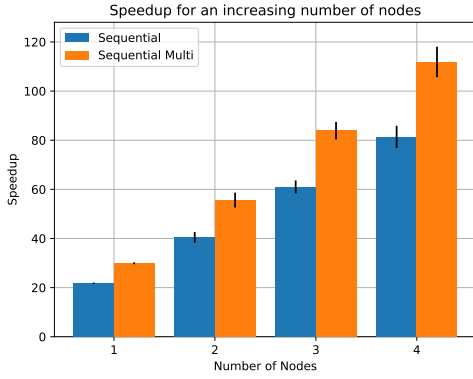
**Figure B.1:** Relative speedup  $S_{rel}$  for an increasing number of evaluations of the INET-LANS model with both the *Sequential*, *Sequential Multi* execution time baselines when SMT is enabled or disabled.

### B.2 Experiment: PDES

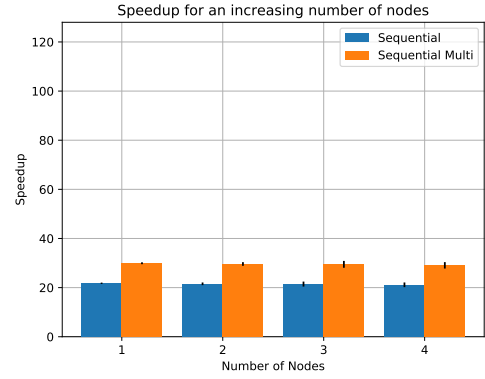
All additional figures belonging to the PDES experiment are presented in this section.



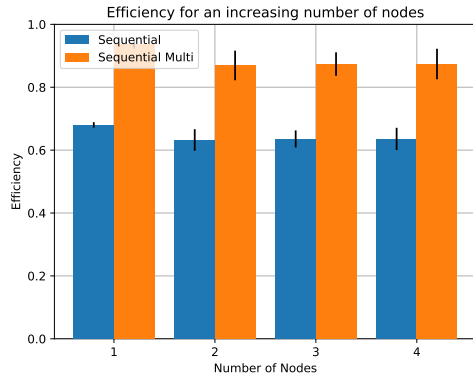
### B.2.1 Sequential Campaign



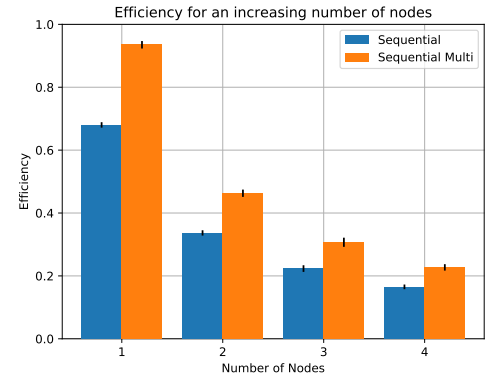
(a) Speedup for weak scaling



(b) Speedup for strong scaling



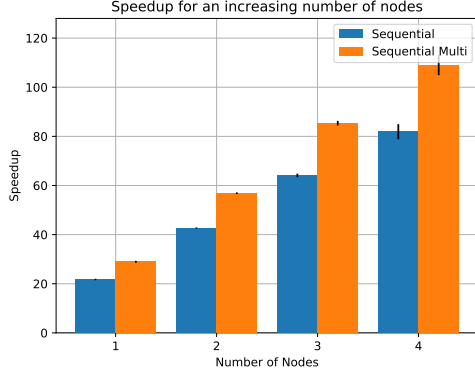
(c) Efficiency for weak scaling



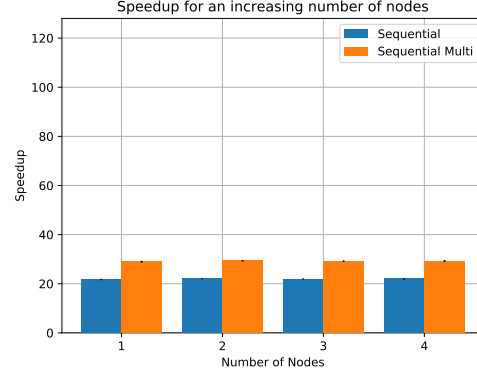
(d) Efficiency for strong scaling

**Figure B.2:** Efficiency and Speedup for strong and weak scaling with 32 sequential evaluations of the communication intensive CQN model as baseline for the single node configuration and the *Sequential* and *Sequential Multi* execution time baselines.

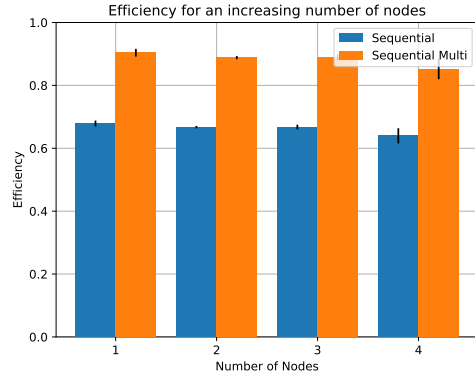
## B. VISUALISATIONS



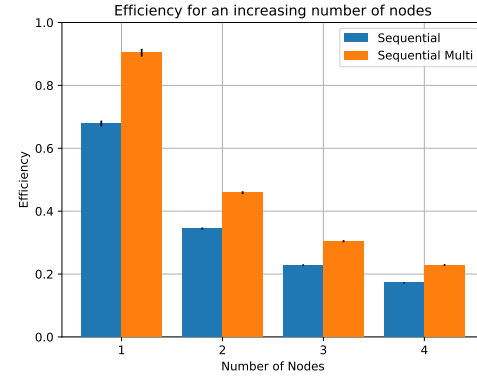
(a) Speedup for weak scaling



(b) Speedup for strong scaling



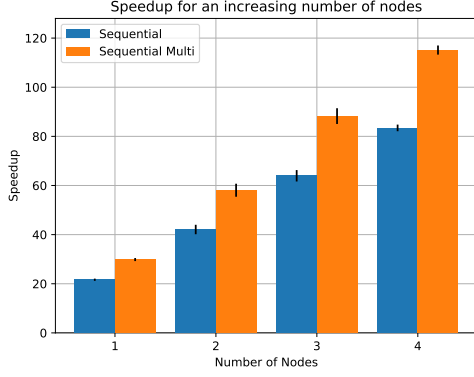
(c) Efficiency for weak scaling



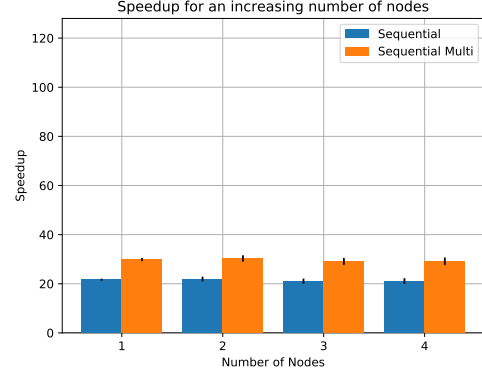
(d) Efficiency for strong scaling

**Figure B.3:** Efficiency and Speedup for strong and weak scaling with 32 sequential evaluations of the computation intensive CQN model as baseline for the single node configuration and the *Sequential* and *Sequential Multi* execution time baselines.

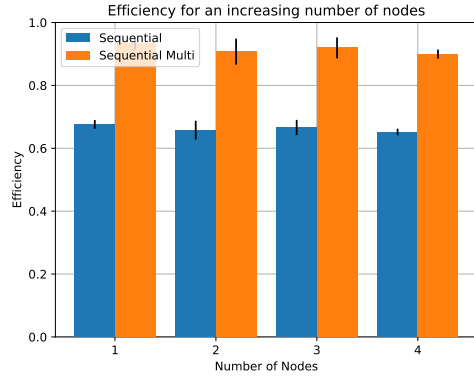
## B.2 Experiment: PDES



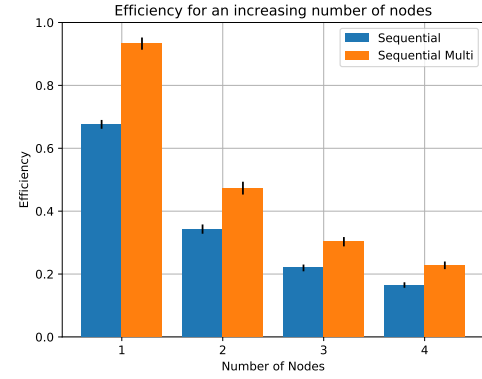
(a) Speedup for weak scaling



(b) Speedup for strong scaling



(c) Efficiency for weak scaling



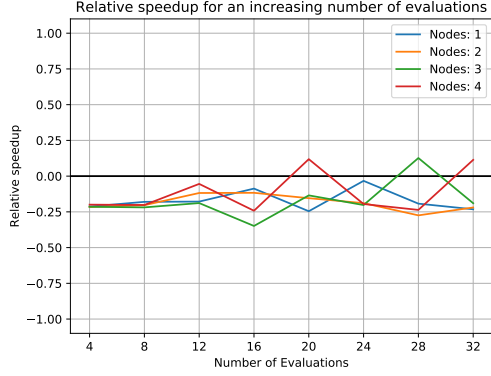
(d) Efficiency for strong scaling

**Figure B.4:** Efficiency and Speedup for strong and weak scaling with 32 sequential evaluations of the subsystem intensive CQN model as baseline for the single node configuration and the *Sequential* and *Sequential Multi* execution time baselines.

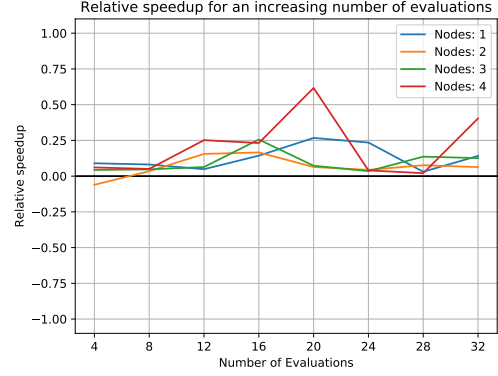
## B. VISUALISATIONS

---

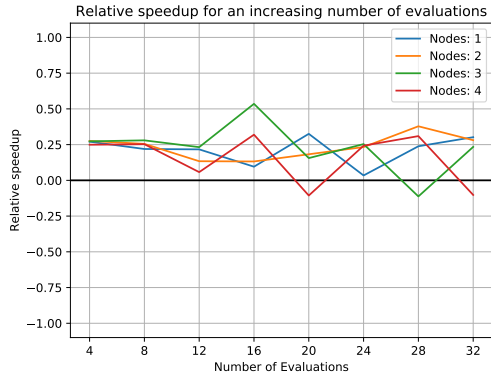
## B.2.2 Relative Speedup



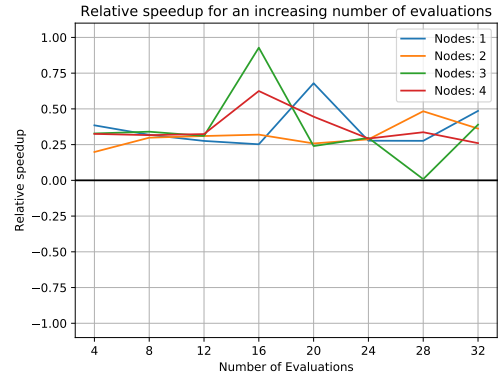
(a) Relative speedup for computation intensive against communication intensive ( $S_{rel}(comp, comm)$ ) with *Sequential* baseline



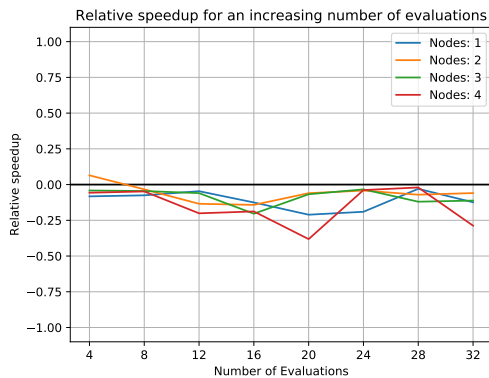
(b) Relative speedup for subsystem intensive against communication intensive ( $S_{rel}(sub, comm)$ ) with *Sequential* baseline



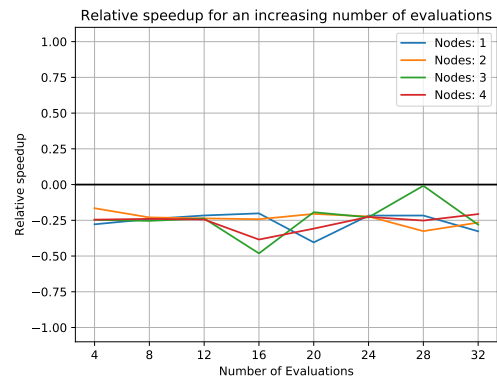
(c) Relative speedup for communication intensive against computation intensive ( $S_{rel}(comm, comp)$ ) with *Sequential* baseline



(d) Relative speedup for subsystem intensive against computation intensive ( $S_{rel}(sub, comp)$ ) with *Sequential* baseline



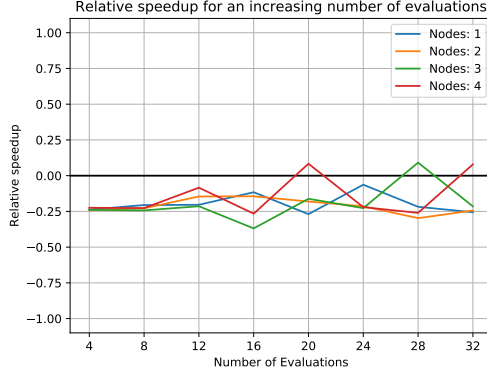
(e) Relative speedup for communication intensive against subsystem intensive ( $S_{rel}(comm, sub)$ ) with *Sequential* baseline



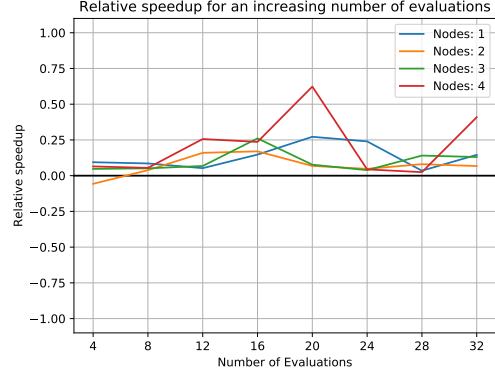
(f) Relative speedup for computation intensive against subsystem intensive ( $S_{rel}(comp, sub)$ ) with *Sequential* baseline

**Figure B.5:** Relative speedup  $S_{rel}$  for increasing number of PDES CQN model evaluations with the *Sequential* execution time baseline.

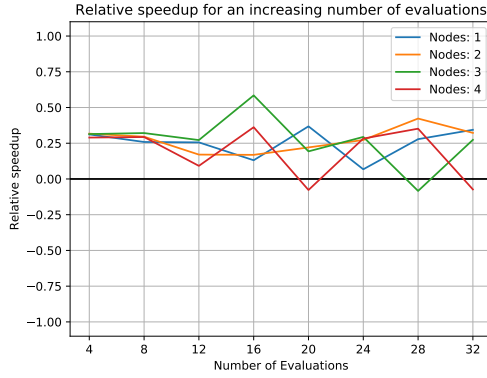
## B. VISUALISATIONS



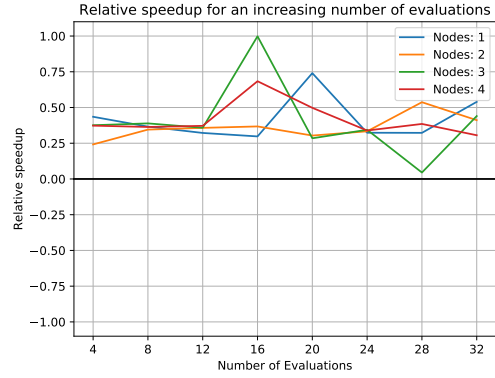
(a) Relative speedup for computation intensive against communication intensive ( $S_{rel}(comp, comm)$ ) with *Sequential Multi* baseline



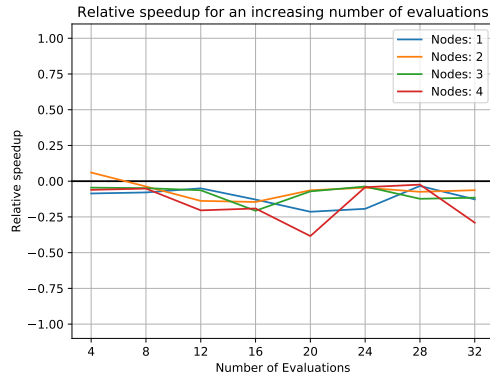
(b) Relative speedup for subsystem intensive against communication intensive ( $S_{rel}(sub, comm)$ ) with *Sequential Multi* baseline



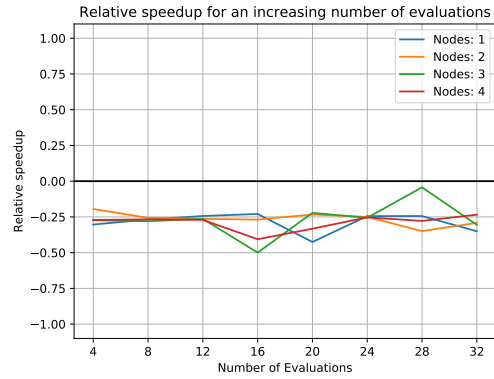
(c) Relative speedup for communication intensive against computation intensive ( $S_{rel}(comm, comp)$ ) with *Sequential Multi* baseline



(d) Relative speedup for subsystem intensive against computation intensive ( $S_{rel}(sub, comp)$ ) with *Sequential Multi* baseline



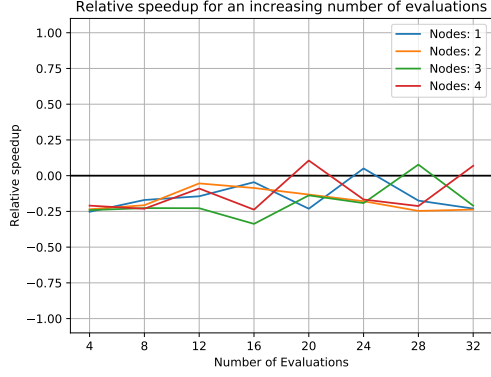
(e) Relative speedup for communication intensive against subsystem intensive ( $S_{rel}(comm, sub)$ ) with *Sequential Multi* baseline



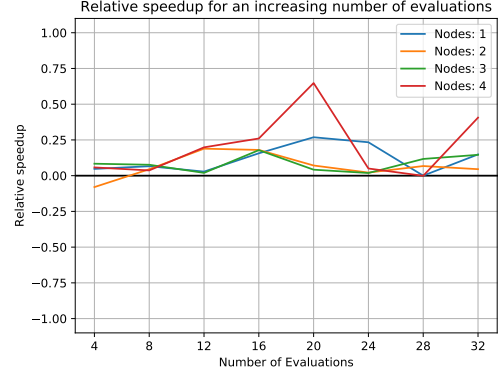
(f) Relative speedup for computation intensive against subsystem intensive ( $S_{rel}(comp, sub)$ ) with *Sequential Multi* baseline

**Figure B.6:** Relative speedup  $S_{rel}$  for increasing number of PDES CQN model evaluations with the *Sequential Multi* execution time baseline.

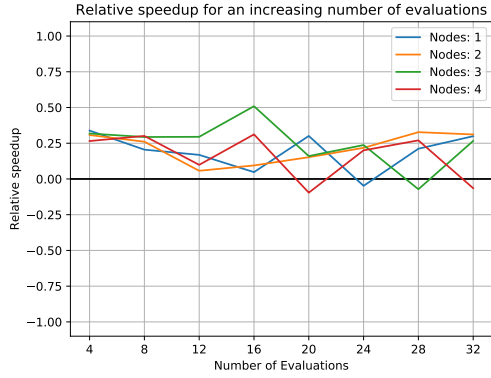
## B.2 Experiment: PDES



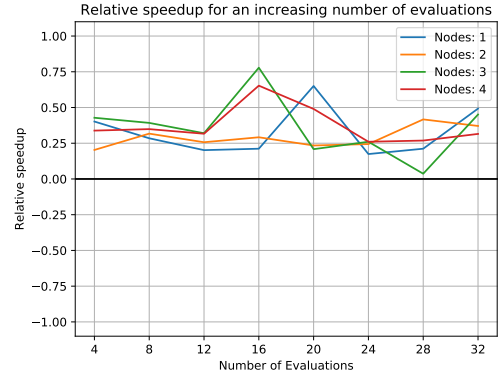
(a) Relative speedup for computation intensive against communication intensive ( $S_{rel}(comp, comm)$ ) with *Sequential Campaign* baseline



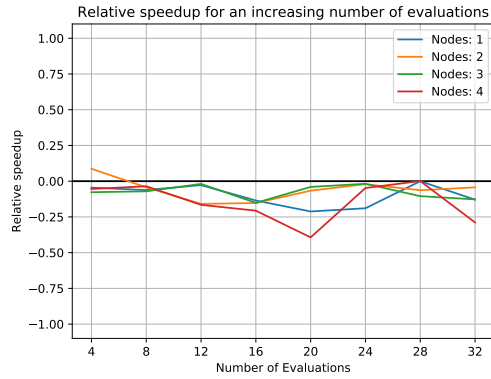
(b) Relative speedup for subsystem intensive against communication intensive ( $S_{rel}(sub, comm)$ ) with *Sequential Campaign* baseline



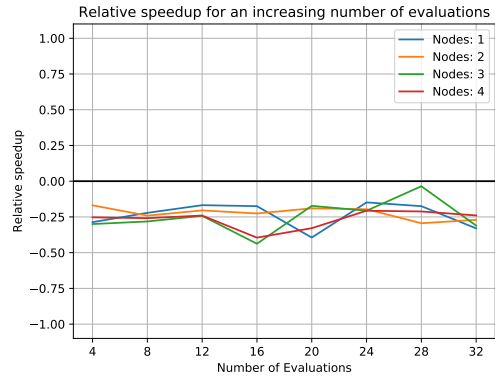
(c) Relative speedup for communication intensive against computation intensive ( $S_{rel}(comm, comp)$ ) with *Sequential Campaign* baseline



(d) Relative speedup for subsystem intensive against computation intensive ( $S_{rel}(sub, comp)$ ) with *Sequential Campaign* baseline



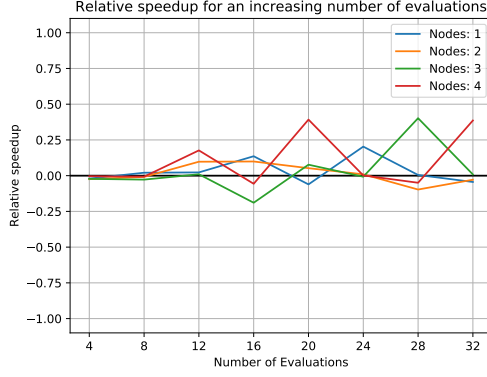
(e) Relative speedup for communication intensive against subsystem intensive ( $S_{rel}(comm, sub)$ ) with *Sequential Campaign* baseline



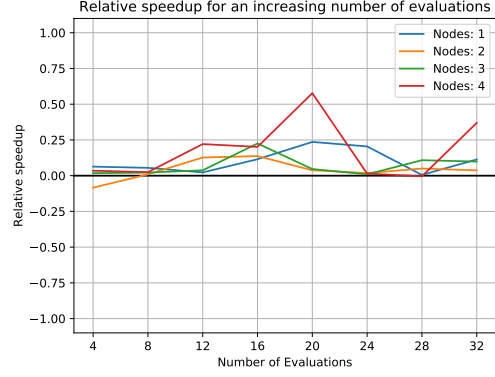
(f) Relative speedup for computation intensive against subsystem intensive ( $S_{rel}(comp, sub)$ ) with *Sequential Campaign* baseline

**Figure B.7:** Relative speedup  $S_{rel}$  for increasing number of PDES CQN model evaluations with the *Sequential Campaign* execution time baseline.

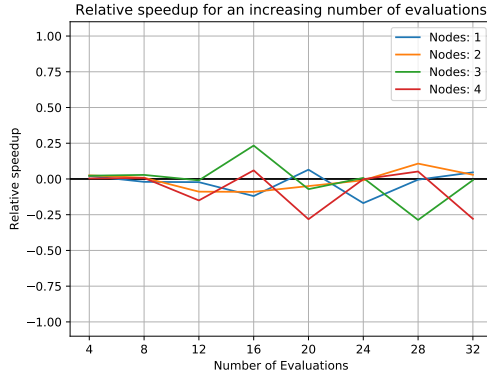
## B. VISUALISATIONS



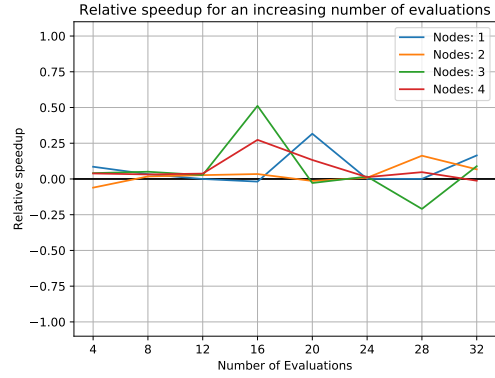
(a) Relative speedup for computation intensive against communication intensive ( $S_{rel}(comp, comm)$ ) with *Sequential PDES* baseline



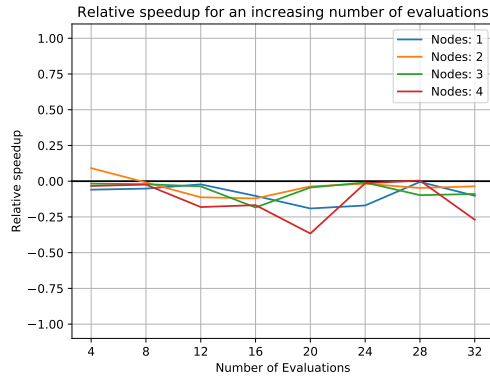
(b) Relative speedup for subsystem intensive against communication intensive ( $S_{rel}(sub, comm)$ ) with *Sequential PDES* baseline



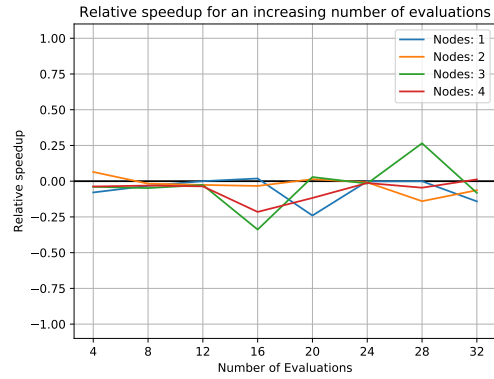
(c) Relative speedup for communication intensive against computation intensive ( $S_{rel}(comm, comp)$ ) with *Sequential PDES* baseline



(d) Relative speedup for subsystem intensive against computation intensive ( $S_{rel}(sub, comp)$ ) with *Sequential PDES* baseline



(e) Relative speedup for communication intensive against subsystem intensive ( $S_{rel}(comm, sub)$ ) with *Sequential PDES* baseline

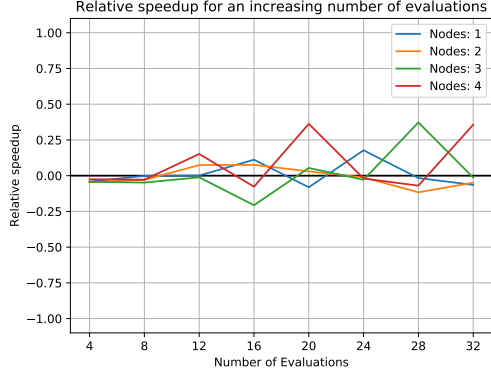


(f) Relative speedup for computation intensive against subsystem intensive ( $S_{rel}(comp, sub)$ ) with *Sequential PDES* baseline

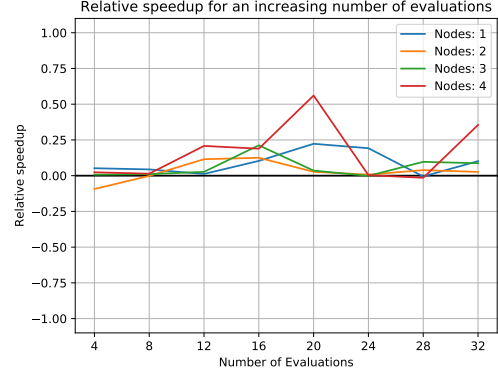
**Figure B.8:** Relative speedup  $S_{rel}$  for increasing number of PDES CQN model evaluations with the *Sequential PDES* execution time baseline.



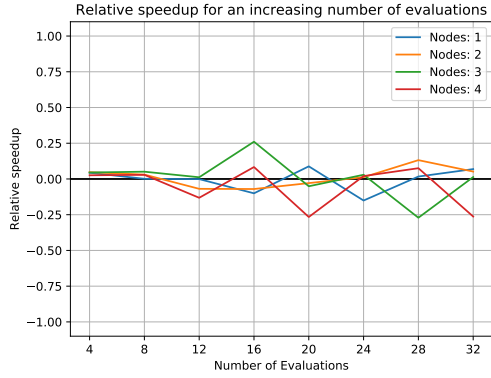
## B.2 Experiment: PDES



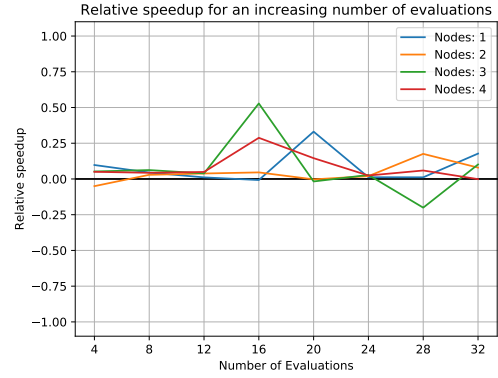
(a) Relative speedup for computation intensive against communication intensive ( $S_{rel}(comp, comm)$ ) with *Sequential PDES Multi* baseline



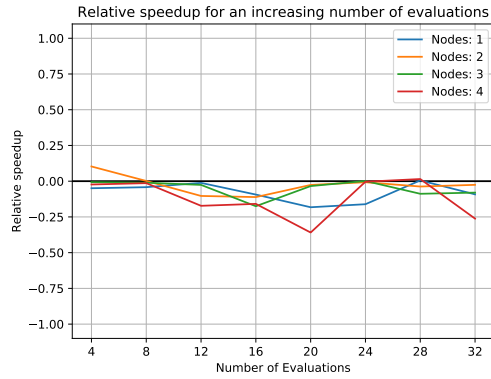
(b) Relative speedup for subsystem intensive against communication intensive ( $S_{rel}(sub, comm)$ ) with *Sequential PDES Multi* baseline



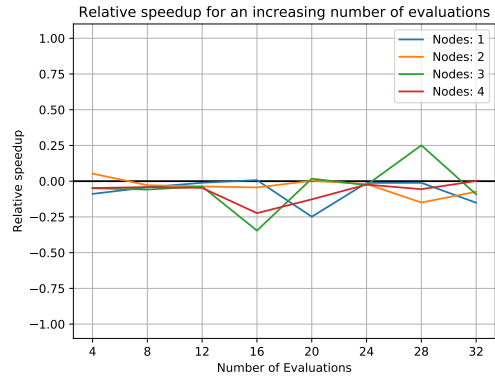
(c) Relative speedup for communication intensive against computation intensive ( $S_{rel}(comm, comp)$ ) with *Sequential PDES Multi* baseline



(d) Relative speedup for subsystem intensive against computation intensive ( $S_{rel}(sub, comp)$ ) with *Sequential PDES Multi* baseline



(e) Relative speedup for communication intensive against subsystem intensive ( $S_{rel}(comm, sub)$ ) with *Sequential PDES Multi* baseline



(f) Relative speedup for computation intensive against subsystem intensive ( $S_{rel}(comp, sub)$ ) with *Sequential PDES Multi* baseline

**Figure B.9:** Relative speedup  $S_{rel}$  for increasing number of PDES CQN model evaluations with the *Sequential PDES Multi* execution time baseline.



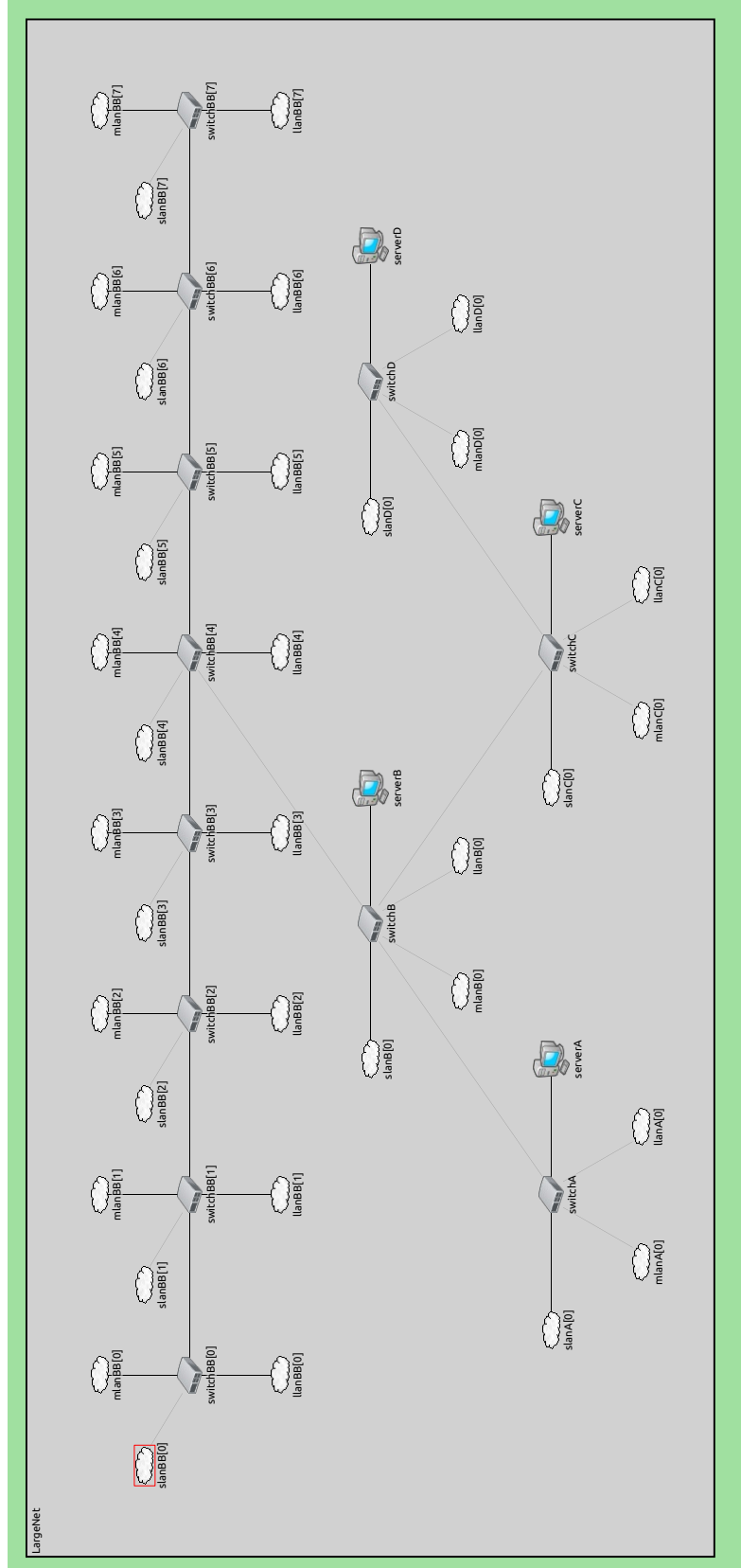
## Appendix C

### Models

The Models appendix contains additional figures representing models used throughout the evaluation, but not presented (or presented differently) in Chapter 5.

#### C.1 INET-LANS

All additional figures belonging to the INET-LANS model are presented in this section.



**Figure C.1:** Visualisation of the INET-LANS model. Each LAN edge-node in the graph, visualised by a cloud, contains a small, medium, or large LAN with hosts connected to a switch and hub.