FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Predictive Approaches for Resource Provisioning in Distributed Systems

DISSERTATION

zur Erlangung des akademischen Grades

Doktor der Technischen Wissenschaften

eingereicht von

Dipl.-Ing. Michael Borkowski, BSc
Matrikelnummer 00925853

an der Fakultät für Informatik
der Technischen Universität Wien
Betreuung: Associate Prof. Dr.-Ing. Stefan Schulte

Diese Dissertation haben begutachtet:

Valeria Cardellini	Stefan Tai

Wien, 1. März 2020

Michael Borkowski

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Predictive Approaches for Resource Provisioning in Distributed Systems

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

Dipl.-Ing. Michael Borkowski, BSc
Registration Number 00925853

to the Faculty of Informatics

at the TU Wien

Advisor: Associate Prof. Dr.-Ing. Stefan Schulte

The dissertation has been reviewed by:

Valeria Cardellini	Stefan Tai

Vienna, 1st March, 2020

Michael Borkowski

Erklärung zur Verfassung der Arbeit

Michael Borkowski
Lilienthalplatz 7
38108 Braunschweig
Deutschland

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Braunschweig, 1. März 2020

Michael Borkowski

Abstract

Modern distributed systems, such as cloud computing infrastructures or data stream processing engines, perform resource provisioning tasks such as resource allocation, task scheduling, or scaling. This decision-making substantially influences the systems' performance, and therefore, the manner of reaching these decisions is crucial to the systems' operation with regard to cost efficiency, performance, reliability, and adherence to service level agreements.

Currently, many approaches to resource provisioning in distributed systems are reactive, i.e., they measure the systems' state, analyze it, and perform necessary actions. The main downside of reactive approaches is that effectively, such systems perform resource provisioning based only on past observations. In a highly dynamic environment with rapidly changing demands for computational resources, this can lead to delayed reactions, which increase cost, degrade performance, and reduce reliability.

This thesis proposes the use of predictive technologies for performing resource provisioning tasks in modern distributed systems. As a foundation, methods stemming from research in the field of machine learning are used to improve target metrics like system performance or operational cost. In contrast to traditional, reactive approaches, the proposed methodology of predictive decision-making is able to perform operational tasks ahead of time, such as scaling out in advance for a predicted increase of demand.

We show how to use predictive methods in various domains of distributed systems, namely cloud computing, business process management systems, data stream processing, and blockchains. We propose approaches to solving challenges in designing predictive methods, such as metric prediction, failure prediction, or data filtering and estimation. We evaluate the impact of the proposed methods on the system using various quantitative methods, including testbed evaluation and simulation, as well as formal and qualitative analysis. Our results show that employing predictive approaches in these domains of distributed systems significantly improves performance attributes such as response time or adherence to service level agreements.

Kurzfassung

Verteilte Systeme, wie sie heutzutage in Bereichen wie Cloud Computing und Data Stream Processing verwendet werden, erfüllen Aufgaben wie Ressourcenzuweisung, Aufgabenplanung und Skalierung. Die hierbei getroffenen Entscheidungen beeinflussen maßgeblich die Systemleistung und sind dementsprechend entscheidend für die Betriebskosten, die Leistungsfähigkeit und die Ausfallsicherheit eines Systems sowie dessen Einhaltung von Service Level Agreements.

Viele aktuelle Techniken zur Ressourcenzuweisung sind reaktiv, überwachen also den Systemzustand und reagieren mit entsprechenden Maßnahmen auf dessen Änderungen. Da solche Ansätze lediglich auf bereits vergangene Messungen zurückgreifen können, kann dies in dynamischen Umgebungen zu maßgeblichen Verzögerungen in der Reaktionszeit, verminderter Leistung, erhöhter Ausfallwahrscheinlichkeit und folglich zu erhöhten Kosten führen.

Diese Arbeit behandelt vorhersagebasierte Techniken zur Ressourcenzuweisung in verteilten Systemen. Hierbei werden Methoden aus dem Gebiet des maschinellen Lernens eingesetzt, um Zielvariablen wie Leistung oder Kosten zu verbessern. Die in dieser Arbeit präsentierten Ansätze ermöglichen es, Maßnahmen zur zeitgerechten Vorbereitung auf wahrscheinliche Zustandsänderungen zu setzen. So können etwa zusätzliche Ressourcen bereits aktiviert werden, bevor sich die Systemlast erhöht.

Vorhersagebasierte Techniken werden in dieser Arbeit in verschiedenen Domänen eingesetzt, nämlich Cloud Computing, Business Process Management, Data Stream Processing und Blockchains. Konkrete Ansätze werden behandelt, wie das Vorhersagen von Metriken und Fehlern, das Filtern von Daten sowie deren Abschätzung. Die Auswirkungen auf das zugrundeliegende System werden experimentell mittels quantitativer Methoden sowie formal und qualitativ analysiert. In den Ergebnissen zeigt sich, dass die Ansätze einen positiven Effekt auf die Zielvariablen haben, wie etwa eine Senkung der Kosten, eine bessere Einhaltung von Service Level Agreements oder eine Minderung von Antwortzeiten.

Danksagung

Der Abschluss meines Doktoratsstudiums, welches in dieser Arbeit gipfelt, stellt für mich einen wichtigen persönlichen Meilenstein dar. Ich möchte mich bei meinen Eltern Irena und Tomasz sowie bei meiner Schwester Barbara bedanken. Meine Familie hat mich während meines gesamten Studiums unterstützt.

Auf universitärer Seite gebührt mein tiefster und aufrichtigster Dank und Respekt meinem Doktorvater, Associate Prof. Dr.-Ing. Stefan Schulte – ein Doktorat steht und fällt mit einer guten Beziehung zwischen Betreuer und Doktorand. Ich kann mich besonders glücklich schätzen, von ihm eine derart lehrreiche Unterstützung in akademischen, professionellen und persönlichkeitsbildenden Belangen erhalten zu haben, und bin dankbar für alle Erfahrungen und Lernwerte, die ich mitnehmen konnte. Univ.-Prof. Dr. Schahram Dustdar möchte ich für die Möglichkeit herzlich danken, mein Doktorat an der DSG abzuschließen.

Herzlichst bedanken möchte ich mich auch bei meinen Kollegen, insbesondere Philipp, Christoph, Svetoslav, Olena und Matteo. Unsere gemeinsame Forschung, Projektarbeit, konstruktive Diskussionen und eure Gesellschaft – sei es am Institut, auf Dienstreisen oder privat im Downstairs – waren ein elementarer Bestandteil meines Forschungsalltags und haben mich stets in meinem Tun motiviert. Meine Arbeit wäre außerdem nicht dieselbe ohne die stetige kulinarische Versorgung durch das Team der Gorilla Kitchen.

Glücklicherweise konnte ich auch außerhalb der TU Wien immer auf Unterstützung seitens meiner Freunde zählen. Besonders dankbar bin ich Manuela, Barbara, Sophie und Simone, sowie Christian, Konrad und Marco. Ihr habt mich in guten wie in schlechten Zeiten immer unterstützt und ermutigt. Dank Stefanie und Elisabeth fiel es mir schwer, im April 2019 Wien zu verlassen, dank Max, Joonas und Marc sowie der Nordstadt-Gang Eli, Inki, Nele, Lena und Ivo allerdings umso leichter, in der Löwenstadt Fuß zu fassen.

Schlussendlich möchte ich meine Anerkennung, meinen Respekt und meine Dankbarkeit gegenüber der gesamten akademischen Gesellschaft zum Ausdruck bringen. Studenten, Doktoranden, wissenschaftliche Mitarbeiter, Professoren, genauso wie Gutachter, Herausgeber von wissenschaftlichen Fachzeitschriften und Organisatoren von Konferenzen ermöglichen durch kontinuierliche, gewissenhafte, harte [76, 251] und trotzdem oft zu wenig wertgeschätzte Forschungsarbeit Fortschritte, welche unseren Horizont erweitern.

Acknowledgements

I am very grateful to Associate Prof. Dr. Valeria Cardellini of the University of Rome Tor Vergata, as well as Prof. Dr.-Ing. Stefan Tai of TU Berlin for agreeing to serve as reviewers for this thesis and providing me with very valuable input.

I would like to thank Christoph, Daniel, Jon, Oskar, and Taneli, who—despite my Java background—welcomed me and let me become a member of Team Python at heart. Thank you for the fruitful and interesting discussions, for the funny remarks, calculations of Erdősz numbers, linguistic discourses, delightful lunch and coffee breaks, and for the overall pleasant working environment, allowing me to have a good time at Bitpanda.

Additionally, my thanks go out to my friend Pablo Hofbauer of the Institute of Molecular Biotechnology (IMBA) of the Austrian Academy of Sciences (ÖAW) for kindly providing the dataset used in the evaluation of Chapter 5, and for continuously offering insights into research techniques in biomedical engineering.

Finally, I would like to thank the scientific community, including all authors and co-authors, students, advisors, professors, editors, reviewers, grant sponsors, chairs, as well everyone involved in organizing venues such as conferences and workshops, for their efforts, criticism, ideas, input, and support. Research is hard work [76, 251], often voluntary, unpaid and mostly underappreciated, even though it results in a significant amount of progress in science. I therefore especially acknowledge the work done by my fellow PhD students in all fields. Together, we can continue broadening horizons.

The work presented in this thesis is supported by the Commission of the European Union within the CREMA H2020-RIA project (Grant agreement no. 637066), by Pantos GmbH within the TAST research project, by the Vienna Science and Technology Fund (WWTF) through project ICT15-072, by the Österreichische Forschungsförderungsgesellschaft (FFG) through the Austrian Center for Digital Production, by the TU Wien University Library, and by TU Wien research funds. The original idea to the work presented in Chapter 4 is a result of the GI-Dagstuhl Seminar 16341 “Integrating Process-Oriented and Event-Based Systems”. Furthermore, the author is thankful for having received support by the Institute of Flight Guidance at the German Aerospace Center (DLR, FL-PAS) in Braunschweig, Germany.

To all my dear friends.

Contents

List of Figures

List of Tables

Acronyms

ANN Artificial Neural Network

API Application Programming Interface

ARIMA Autoregressive Integrated Moving Average

ARMA Autoregressive Moving Average

AWS Amazon Web Services

BAM Business Activity Monitoring

BPI Business Process Intelligence

BPIC Business Process Intelligence Challenge

BPM Business Process Management

BPMN Business Process Model and Notation

BPMS Business Process Management System

CBT Cross-Blockchain Token

CDF Cumulative Distribution Function

CEP Complex Event Processing

CFS Commodity Flow Survey

CI Continuous Integration

CPEE Cloud Process Execution Engine

CPU Central Processing Unit

DeXTT Decentralized Cross-Blockchain Token Transfers

DSP Data Stream Processing

EBS Event-Based System

ECDSA Elliptic Curve Digital Signature Algorithm

EDA Event-Driven Architecture

EFP Event-Based Failure Prediction

EKF Extended Kalman Filter

EVM Ethereum Virtual Machine

FFT Fast Fourier Transform

FIFO First In, First Out

FN False Negative

FP False Positive

GA Genetic Algorithm

GPGPU General-Purpose Graphics Processing Unit

GW Generalized Weierstrass

I/O Input/Output

IaaS Infrastructure as a Service

IoT Internet of Things

KF Kalman Filter

KVM Kernel-Based Virtual Machines

LAN Local Area Network

LS Linear Smoothing

LSTM Long Short-Term Memory

MAPE Monitor, Analyze, Plan, Execute

MCC Matthew's Correlation Coefficient

MILP Mixed Integer Linear Programming

ML Machine Learning

MPC Model-based Predictive Control

NAG Nesterov's Accelerated Gradient

NFA Non-Deterministic Finite Automaton

OLS Ordinary Least Squares

OS Operating System

PA Probabilistic Automaton

PaaS Platform as a Service

PM Physical Machine

PoA Proof of Authority

PoI Proof of Intent

PoS Proof of Stake

PoW Proof of Work

QoS Quality of Service

RMSD Root-Mean-Square Deviation

SaaS Software as a Service

SGD Stochastic Gradient Descent

SLA Service Level Agreement

SPE Stream Processing Engine

SVM Support Vector Machine

TIFF Tagged Image File Format

TN True Negative

TP True Positive

TVD Total Variation Denoising

UIA User-Issued Asset

USD United States Dollar

VM Virtual Machine

XES Extensible Event Stream

XML Extensible Markup Language

XOR Exclusive Or

XPP Cross-Blockchain Proof Problem

Core Publications

This thesis is based on previously published work. For the sake of brevity and readability, these core papers are listed here once, and are not explicitly referenced throughout the thesis. All papers have been peer-reviewed during the publishing process. The relationship between the individual publications is described in Section 1.3. Parts of the manuscripts are contained in verbatim.

- Michael Borkowski, Marten Sigwart, Philipp Frauenthaler, Taneli Hukkinen, and Stefan Schulte. "DeXTT: Deterministic Cross-Blockchain Token Transfers". In: *IEEE Access* 7.1 (2019), pp. 111030–111042. DOI: 10.1109/ACCESS.2019.2934707

- Michael Borkowski, Christoph Hochreiner, and Stefan Schulte. "Minimizing Cost by Reducing Scaling Operations in Distributed Stream Processing". In: *PVLDB* 12.7 (2019), pp. 724–737. DOI: 10.14778/3317315.3317316

- Michael Borkowski, Walid Fdhila, Matteo Nardelli, Stefanie Rinderle-Ma, and Stefan Schulte. "Event-Based Failure Prediction in Distributed Business Processes". In: *Information Systems* 81 (2019), pp. 220–235. DOI: 10.1016/j.is.2017.12.005

- Michael Borkowski, Stefan Schulte, and Christoph Hochreiner. "Predicting Cloud Resource Utilization". In: *9th IEEE/ACM International Conference on Utility and Cloud Computing (UCC)*. IEEE/ACM, 2016, pp. 37–42. DOI: 10.1145/2996890.2996907

- Michael Borkowski, Christoph Hochreiner, and Stefan Schulte. "Moderated Resource Elasticity for Stream Processing Applications". In: *Parallel Processing Workshops (Euro-Par)*. LNCS 10659. Springer, 2017, pp. 5–16. DOI: 10.1007/978-3-319-75178-8_1

CHAPTER 1

Introduction

Distributed systems play a crucial role in many aspects of today's digital infrastructure, enabling technologies such as cloud storage [78, 111], smart cities [210], or the Internet of Things (IoT) [39]. Research within distributed systems entails fields such as cloud computing [9, 255], Business Process Management (BPM) [89, 232], Data Stream Processing (DSP) [54, 55], or decentralized consensus technologies like blockchains [278]. Common goals within the research field of distributed systems often include increased elasticity, flexibility, and scalability for businesses running a software platform, while maintaining a defined set of cost [232] or quality constraints [53]. Modern distributed systems feature a high degree of complexity, inherent to their distributed architecture. The components necessary to run and maintain such systems form a complex and opaque landscape, which can also be very heterogeneous [178], since these components are often provided by different stakeholders [197], and are executed on diverse hardware using varying technologies [226]. All of this provides the research field of distributed systems with many opportunities for optimization on various levels, and for various metrics.

Elasticity

The process of resource provisioning, i.e., supplying consumers with a set of resources [60, 178], is one of the most researched areas in distributed systems [27, 118]. Often, we consider the problem of how to place applications on Virtual Machines (VMs), or VMs on Physical Machines (PMs) [52, 177]. This process is also called allocation [17]. However, since placement happens on various levels, in more general terms, we use the terminology of placing *tasks* on *resources*. The resource on which a task is placed is occupied for a certain amount of time and to a certain degree, and therefore, the exact assignment between resources and tasks has significant influence on the total resource utilization, affecting the operational cost and performance of the entire system [60].

Resource Provisioning

Scaling

In addition to finding a proper placement, contemporary distributed systems must also maintain elasticity [81, 178], which is achieved by dynamically adapting to workload changes, activating and passivating resources in an autonomic manner. This capability is referred to as *scalability* [122]. The decision of when and how to scale—the *scaling decision*—is a non-trivial problem. Often, there are multiple, conflicting optimization target metrics [67]. For instance, a cloud computing provider is interested in reducing the total cost of operation of the infrastructure, but at the same time, wishes to maintain a given Quality of Service (QoS) to fulfill certain Service Level Agreements (SLAs) in order to avoid paying penalties [53].

While approaches to these tasks are manifold, they usually follow the Monitor, Analyze, Plan, Execute (MAPE) loop [155, 178]. A crucial classification of approaches to resource provisioning is the distinction between *reactive* and *predictive* approaches [50], i.e., approaches which monitor the current system state and react to certain events, and approaches which predict the future development of the system state and proactively take measures to ensure its desired performance [199].

Research Goals

In this thesis, we investigate predictive approaches to resource provisioning in distributed systems. We seek to lay fundamental groundwork for predictive approaches, such as the methods and tools usable for performing the necessary predictions, we show how to apply these methods to problems and use cases found in contemporary distributed systems, and we evaluate their performance. We consider various types of distributed systems found in literature as well as in practice, such as cloud computing [37], Business Process Management Systems (BPMS) [31], DSP [32, 33], and blockchain technologies [38]. We investigate various aspects of predictive approaches, namely the prediction of metric values and their filtering, the prediction of process failures, as well as the employment of determinism to avoid uncertainty. As a result, we seek to gain insight into how various types of use cases, with various types of available data and target systems, can be addressed using predictive approaches.

Methods and Tools

In our proposed approaches to predictive resource provisioning, we use techniques found in the field of Machine Learning (ML), such as Artificial Neural Networks (ANNs) [119]. The variety of system types, scenarios, use cases, and client requests poses a challenge when applying ML techniques. For instance, special care must be taken during the design of ML models, data preparation and imputation, strategies and parameters for training, verification of performance, and integration in operational systems [178].

We show how to address these challenges, how to determine suitable choices of parameters and algorithms, and evaluate the proposed approaches using various quantitative methods, including testbed evaluation and simulation, as well as formal and qualitative analysis. Our results show that employing predictive

approaches in the presented domains of distributed systems can significantly improve performance attributes such as response time or adherence to SLAs.

We propose the following example scenario to illustrate how, instead of using a reactive approach, predictive scaling can be used to maintain elasticity.

> **Example Scenario.** A provider of cloud-based computational services offers a selection of VMs to customers. A customer can lease a VM, use it for hosting applications or performing computations, and release it afterwards. The provider uses an infrastructure consisting of PMs to host customer-leased VMs. In order to maintain economical viability, the provider uses an elastic infrastructure, which automatically performs tasks such as scaling (activating or passivating VMs as necessary), placement (deciding which PM to use for a leased VM), or resource provisioning (deciding how much of which resources—such as storage or network bandwidth—to assign to a VM). Since VMs are used by clients in a dynamic way, and can be leased and released on demand, the infrastructure is subject to dynamic load changes. If VMs are requested by a high number of clients, the infrastructure is under heavier load than during times of low demand.
>
> The provider faces this dynamic environment using predictive approaches: Instead of merely waiting for changes (e.g., the system load exceeding a fixed threshold) and reacting to them, the infrastructure actively predicts whether an increase or decrease of demand is expected. For instance, if an increase is predicted, PMs are proactively activated in order to react to an increased load in a timely manner. Compared to a traditional, reactive approach, such methods improve the system's capabilities to adapt to a changing environment.

1.1 Problem Statement

Designing predictive approaches for distributed systems inherently starts with making predictions about future development of metrics of interest, and as such, requires data as basis for such predictions. A popular example of a source of data is found in historical system observation [156]. Once data is selected and obtained, we require tools and methods for making precise forecasts about the values of metrics in the future, such as in the next minutes, hours, or days, depending on the use case. In many contemporary approaches, this is either not done at all (reactive approaches) [178], or relatively simple methods like linear regression are used [135], yielding mediocre results.

As an overarching goal of this thesis, we seek to determine how to design predictive approaches based on the application scenario. Different environments and different objectives in a given use case require different solutions with specialized tools and techniques.

A prime class of tools and candidate for finding methods to predicting system state is ML [100]. While ML provides a rich set of tools suitable for this purpose, the variety of types of ML models, together with numerous variations and possible modifications or extensions, poses a challenge for the design process of predictive approaches, i.e., for the decision of which technique to use based on the scenario at hand. This is especially the case for distributed systems, where attention must be paid to efficiency and scalability [9].

In addition, ML models usually have a high number of parameters and hyper-parameters, and their selection is often done on a trial-and-error basis. In current literature in the field of distributed systems, there is a lack of substantial reference for selecting these models and parameters—at times, a ML model is described as a *black box* [47]—and researchers as well as developers must invest time and effort for explorative analysis.

When applying methods from literature in practical settings, practical challenges emerge. While the rich set of tools provided by ML for processing data—once a selection of such tools is made—is suitable for creating predictions [181], in practice, data is often incomplete, or noisy [252]. We require techniques to overcome these challenges, either by data pre-processing and imputation, or by adapting mechanisms to filter incomplete data while maintaining functionality.

Finally, in some scenarios, prediction of a certain metric can simply be impossible due to reasons outside of the designer's control. This can be due to non-recurring events and very extraordinary circumstances. We seek ways of dealing with these scenarios in unconventional ways, for instance, by changing the context of the system, circumventing the prediction problem entirely.

1.2 Research Questions

In the following, we formulate three research questions (referred to as RQ1 through RQ3), which serve as motivation for the work at hand. The research questions provide a guide for the core chapters of this thesis, and will be revisited in Chapter 7, where we assess how our contributions address the objectives defined in each individual research question.

> **Research Question I.** How can predictive techniques be used in distributed systems to optimize operational cost, performance, and reliability?

As described in Section 1.1, ML provides numerous tools and methods for predicting future system state [181]. A key challenge is the application of these methods in the domain of distributed systems, since due to high system complexity, the data collection, aggregation, processing, and finally forecasting, is an

elaborate process. In contrast to centralized applications of ML (e.g., economic forecasting [209], power systems [270], natural language processing [183], or image recognition [157]), applications in highly distributed systems such as cloud computing, BPMS, or DSP, require increased attention to scalability and computational complexity [9].

Special care must be taken to consider these factors throughout the entire process of design, implementation, parameterization, and operation. Furthermore, as discussed above, we aim to take into account various types of scenarios and consider the varying objectives encountered in the various use cases. Based on this, we aim to select techniques and methods suitable for the scenario at hand.

Research Question II. How can predictive approaches maintain functionality and performance in unknown and uncertain situations?

Network connections and their inherent unreliability are ubiquitous in distributed systems. Especially when using contemporary paradigms like fog computing [239], each system component must be tolerant to uncertainty such as network disruptions, increased delay, reduced bandwidth, or total loss of connectivity, i.e., in a volatile environment [254]. During the disruption, the system must maintain operation within the boundaries of possibility, as dictated by the given use case. Once the disruption ceases, the system must commence normal operation. This tolerance implies that a predictive approach must be able to function properly even in times where network connections are disrupted, and therefore metrics are unreliable, imprecise, or unavailable.

Research Question III. Can uncertainty be overcome by adapting the context of the system?

In some scenarios, the future value of a metric is uncertain, and simply cannot be predicted due to insufficient or missing data. In these situations, a possible approach is to reconsider the overall process requirements: Sometimes, the inability to answer a given question can be addressed by asking a different question. We seek to find ways of solving issues of insufficient or missing data by changing the system in a way that makes the system independent of this data.

1.3 Scientific Contributions

Table 1.1: Summary of core thesis contributions

Reference	Contribution	Domain	Method	Main outcome
[37]	Chapter 3	Cloud computing	ANN regression	Error ratio: 0.77
[31]	Chapter 4	EBS/BPMS	ANN classification	Precision: 87%
[32, 33]	Chapter 5	DSP	EKF-based filter	Scaling reduction: 88%
[38]	Chapter 6	Blockchains	Deterministic witnesses [36]	Deterministic result

Guided by the research questions formulated in Section 1.2, the core contributions are highlighted in the following to provide an outline for this thesis. Table 1.1 gives an overview of the core contributions of this thesis, showing the domain of each contribution, the method used, and the main outcome.

Contribution I. Predicting Resource Utilization in Cloud Computing

"Predicting Cloud Resource Utilization" [37]

The first contribution is embedded in the context of cloud computing, and entails an approach to predicting resource utilization in the cloud. The presented approach uses ANNs to train a ML model based on historical data of tasks processed by a cloud computing infrastructure. The resulting ML model is capable of creating resource utilization predictions for future incoming tasks based on their characteristics. The model yields an error ratio of 0.77, i.e., a decrease of 23% in prediction error, compared to the state of the art. Contribution I has originally been presented in [37].

Contribution II. Failure Prediction in Distributed Business Process Management Systems

"Event-based Failure Prediction in Distributed Business Processes" [31]

The second contribution introduces an approach to failure prediction in BPMS. In such systems, knowledge about an imminent or likely failure can be used to re-provision resources, for instance, to start another process, or to activate mitigation strategies. To this end, we connect the two research fields of Event-Based Systems (EBS) and BPMS, allowing us to leverage on the context of a business process and to facilitate this previously-unused source of data.

We show that this process context, which often provides a high-volume stream of events, is a suitable data source for ML training. Our ML model, using an advanced type of ANNs, is able to predict the failure likelihood in a business process. In addition, the prediction also states which process step is the likely point of failure. In our evaluation, we show that the prediction performance is significantly increased by using context events. The resulting ML model yields a

precision of 87%. In addition, we show how to employ failure prediction not only in a local, intra-organizational setting, but also in a distributed scenario, where various stakeholders collaborate in a business process, but do not share the entirety of available data. Contribution II has originally been presented in [31].

Contribution III. Predictive Scaling in Data Stream Processing

The third contribution discusses elasticity in the context of DSP. We show how advanced filtering, such as Total Variation Denoising (TVD) and Extended Kalman Filters (EKFs), can be used to pre-process time series data and filter out unwanted noise. The usage of such filters allows us to estimate the actual state of a system with respect to relevant metrics from noisy and imprecise data, and based on this, perform scaling operations. Our evaluation is conducted using a testbed, where a real-life DSP scenario found in the research field of biomedical engineering [137] is enacted, and performance metrics such as Central Processing Unit (CPU) utilization and SLA violations are observed. Our evaluation shows that while the required CPU utilization increases by up to 13.9% when using our proposed EKF-based approach, the amount of scaling events is reduced by up to 52.4%, and the amount of SLA violations decreases by up to 15.2%. Based on this, we provide a cost discussion and a break-even point analysis to determine the conditions necessary for the EKF-based approach to provide cost benefits. Contribution III has originally been presented in [32, 33].

"Minimizing Cost by Reducing Scaling Operations in Distributed Stream Processing" [32]

"Moderated Resource Elasticity for Stream Processing Applications" [33]

Contribution IV. Determinism as a Response to Uncertainty

The fourth and final contribution deals with uncertainty on a higher level and tackles situations where no predictions are possible due to circumstances like the absence of a suitable data source. We address this challenge by changing the system definition in a way that circumvents the missing data. This contribution is embedded in the context of blockchains, i.e., decentralized consensus, where multiple parties are competing for a reward in a smart contract. It is crucial to anticipate which party will win this competition; however, a priori, there is no means of predicting this outcome.

We therefore redesign the protocol in a way that creates determinism by introducing a metric which is easy to evaluate, but very hard to tamper with, effectively selecting a winning party in a random (in the sense that each party has an equal chance of winning), yet deterministic manner. Contribution IV has originally been presented in [38].

"DeXTT: Deterministic Cross-Blockchain Token Transfers" [38]

1.4 Thesis Structure

The remainder of this thesis is structured as follows. Chapter 2 provides background information, specifically, a discussion of distributed systems (Section 2.1), approaches to maintain elasticity (Section 2.2), techniques for creating predictions (Section 2.3), and ANNs (Section 2.4). In Section 2.5, we highlight the state of the art and discuss gaps which we will address in the remainder of the thesis. Chapters 3 through 6 present the main matter of this thesis, and constitute Contributions I–IV in accordance with Section 1.3. Chapter 3 shows an approach to predicting resource utilization in the context of cloud computing. In Chapter 4, we combine the fields of EBS and BPMS, and show how to perform failure prediction using ML. Chapter 5 demonstrates the usage of EKFs to create state estimations while tolerating noisy data. Chapter 6 shows how to circumvent uncertainty by changing the system to create determinism. Finally, in Chapter 7, we conclude the thesis, discuss the contributions presented in the previous chapters in light of the postulated research questions, and offer an outlook for ongoing and future research.

CHAPTER 2

Background

The concepts presented in this thesis are methods applicable in today's distributed systems, which can be used to enhance the functionality of these systems, reduce their operating cost, or optimize resource utilization. In the following sections, we discuss the background of distributed systems, how scalability and elasticity are realized, and why predictive methods are promising to further enhance these features. In addition, we provide a high-level discussion of contemporary literature with regard to resource provisioning and proactive approaches, and highlight gaps, thus motivating the choice of approaches within the individual contributions presented in the subsequent chapters.

2.1 Distributed Systems

Tanenbaum et al. define a distributed system as "*a collection of independent computers that appears to its users as a single coherent system*" [248]. While this definition was coined in 2001, and distributed systems have evolved substantially since that time, the definition still fits the paradigms used in contemporary distributed systems. In this work, we therefore use this definition of a distributed system, and in more detailed contexts, refer to them simply as *systems*.

Transparency

The advent of distributed systems is owed to the features they enable. These features mainly include resource accessibility, transparency, and scalability. Resource accessibility means that resources provided by a remote computer—for instance, data storage or computational power—can be used by clients regardless of their geographic location [248]. Today, cloud storage providers like Dropbox[1] are a prime example of providing accessibility to vast storage resources for clients such as mobile devices. Transparency refers to hiding some aspects of a system,

[1] `https://www.dropbox.com/`

and we differentiate between various transparency types, such as location transparency (hiding where a resource is located), replication transparency (hiding that a resource is replicated), or failure transparency (hiding the failure and recovery of a resource) [248].

Scalability

Scalability is among the most crucial features of modern distributed systems [9, 148]. A system is described as scalable if it is deployable in a wide range of scales, in terms of user number, data amount, processing rate, number of nodes, and geographical coverage. Generally, small scales are just as important as large scales, and scalability means not just the ability to operate, but to operate efficiently and with adequate QoS, over the given range of configurations [148]. Scalability can be realized in a *horizontal* or *vertical* manner [75]. In horizontally elastic systems, computational units such as VMs are added or removed, and this process is called *scaling out*. In contrast, vertical elasticity involves adding or removing resources from those units, such as CPU cores, memory, or storage [166], a process called *scaling up*. The approaches presented in this thesis are not specific to either of these scaling dimensions and can be used for both horizontal and vertical scaling. Furthermore, there exist numerous implementations and classes of scalability [11], with performance-oriented types such as *load scalability* or *space scalability* being among the most important [29]. Other types include *distance scalability* or *speed scalability*. In this work, when describing scalability, we generally refer to load scalability unless noted otherwise.

Elasticity

A concept related to scalability is *elasticity*. Various definitions of elasticity exist [10, 191], and have been collected into a single definition by Herbst et al., where elasticity is defined as "*the degree to which a system is able to adapt to workload changes by provisioning and deprovisioning resources in an autonomic manner, such that at each point in time the available resources* match *the current demand as closely as possible*" [122]. Therefore, we regard scalability as a *requirement* for elasticity [80], since a system featuring elasticity must be scalable. Elastic systems are often also required to feature self-⋆ properties, such as self-healing, self-protecting, or self-stabilizing, collectively referred to as autonomic computing [127]. Generally, elasticity, including scalability and self-⋆ properties, is a means of ensuring an overall system's QoS [49, 57]. Often, a set of requirements about the system's QoS is collected into a document, representing an agreement between a service provider and a service consumer, referred to as SLAs. In some scenarios, SLAs are provided as legally binding contracts [42], and SLA violations cause penalties for the service provider.

Cluster and Grid Computing

Overall progress of research in hardware and software design has led to the advent of several paradigms in distributed systems. Tanenbaum et al. already describe cluster computing systems. A cluster is a set of tightly coupled computers, connected by means of a high-speed Local Area Network (LAN). In addition, each node runs the same Operating System (OS) [248]. A cluster provides various capabilities to its users—for instance, computational power—and benefits from

the sum of resources of the individual computers. A slightly younger paradigm is grid computing, which, like computing clusters, represents a collective of computers. However, in contrast to clusters, the computers making up the grid are federated, often under different administrative domains, and may be different with regard to hardware and software [248]. Furthermore, since computing grids are operated by various groups, they tend to be connected through the Internet, instead of sharing a common LAN connection. Popular examples of highly distributed computing grids are *SETI@home*[2] and *Folding@home*[3], where participants worldwide provide computing resources for a common goal, in these cases analyzing radio telescope data and protein folding, respectively.

Cloud Computing

Cloud computing [15], a paradigm which emerged after the original definition of distributed systems by Tanenbaum et al., represents the idea of *computing as a utility* [46]. Resources such as computational power or storage are hosted in data centers, and provided to their users on-demand. The entirety of these provided resources is commonly called the *cloud* [9]. The utility-like fashion with which the resources are provided is reflected in simple, predictable cost models suitable for scalability, where running 1,000 servers for one hour causes cost comparable to running one server for 1,000 hours [10]. The benefit of this computing paradigm for the consumer is the lack of upfront investment, which allows rapid and cost-efficient application deployment, easy calculation of cost, and therefore bears the potential for scalability. Furthermore, outsourcing the task of maintaining certain parts of the technology stack to the cloud provider frees resources on the consumer side. An organizational (and legal) separation of concerns enables the consumer to focus on the application itself, instead of also having to manage the infrastructure on which the application is hosted.

Virtualization

A major enabler of cloud computing is virtualization [9], which refers to the abstraction of hardware aspects in order to decouple it from the executed software [195]. While an OS is usually tightly coupled to the executing hardware through drivers and low-level code called firmware, a VM represents an emulated (virtual) hardware environment. This is done to allow portability of software between various types of PMs, possibly in different locations. Furthermore, VMs allow the isolation of applications. Applications usually have specific requirements to the execution environment, including runtime libraries and additional services, possibly with specific version constraints. Running two applications on the same host might be impossible due to a conflict of these requirements and a lack of isolation [79]. Isolating each application's required environment using VMs can address this challenge.

Containers

Virtualization itself can occur on various levels. Traditionally, entire machines are virtualized, resulting in VMs, and each VM runs an entire OS, independent of the physical host and of other VMs. Recently, a more lightweight approach to

[2] `https://setiathome.berkeley.edu/`
[3] `https://foldingathome.org/`

virtualization has gained importance, called *containers* [79, 86], with Docker[4] being a popular implementation. Containers place the cut between non-virtualized and virtualized software higher up in the stack, with major parts of the OS kernel shared between individual containers. As containers must still be isolated from each other, this requires support by the kernel (e.g., using `cgroups` and namespaces in the Linux kernel) [79].

In the case of VMs, the virtualization host is called *hypervisor*, referring to both the PM hosting the VMs, as well as the virtualization management software running on the PM, responsible for managing the hosted VMs. In the case of containers, the nomenclature differs, and for Docker, the host is usually called *container host* [208] or *Docker host* [61].

IaaS, PaaS, SaaS

The term cloud computing is used to refer both to the application delivered as a service (e.g., Dropbox), and to the hardware and systems software in the data centers providing these services. Cloud computing is traditionally grouped into three different levels of provided services [172]. On the lowest level, the cloud provider is responsible for the hardware, including the storage, network, computation units and the virtualization host (e.g., the hypervisor). This is referred to as Infrastructure as a Service (IaaS), and its users are responsible for managing the OS and all software running within it (runtime environment such as middleware and libraries, and application software). A higher service level is Platform as a Service (PaaS), where the cloud provider is responsible for all IaaS components, and in addition, also manages the OS and the required runtime environment (middleware, libraries). The user merely deploys the application software onto the PaaS platform. Finally, Software as a Service (SaaS) refers to the entire technology stack, including the application itself, being provided.

Fundamentally, IaaS and PaaS differ from SaaS in that the former two levels are targeted at technical users, e.g., software developers and providers, by design. In contrast, SaaS is (generally) targeted at a broader audience, since no knowledge of deploying OSs, libraries, or other software is required. Amazon Web Services (AWS)[5] is a prime example of an IaaS and PaaS provider. In contrast, Dropbox is an example for SaaS. Note that the consumer of an IaaS or PaaS service can simultaneously be a provider of a SaaS service. In this case, from the end users point of view, the SaaS provider is responsible for the entire technology stack. However, internally, the responsibility for certain parts such as the infrastructure is delegated to the IaaS or PaaS provider (subcontractor).

The shift from conventional computing in data centers to cloud computing entailed the centralization of tasks and responsibilities. Even though the cloud itself is a distributed system (consisting of a number of PMs), to the user, it is a central point of addressing for the service it provides. For instance, even though

[4] `https://www.docker.com/`
[5] `https://aws.amazon.com/`

Dropbox is a distributed cloud application, a user considers it their central point of storage for data. This shift towards centralization has both benefits and drawbacks. While the main benefits are an increase in availability, performance, and easy cost calculation [9], performing all tasks centralized in the cloud also poses the drawback of higher communication overhead [248]. In some scenarios, the network delay induced by sending data to the cloud for processing, and waiting for a response, is not acceptable, for instance, mobile scenarios, where delays and energy consumption are crucial [165].

Fog Computing

This lack of solutions between on-premise and cloud systems has been the target for the recent development of *fog computing*, named after a cloud close to the ground [30]. Fog computing describes a paradigm filling the gap between on-premise (edge) and cloud computing [20], where devices physically close to the edge of the network provide some of the required resources (e.g., computing, storage), while the cloud works as a supplementary or fallback resource, depending on the implementation. Due to this fact, the term *edge computing* is also prevalent, and fog computing is generally understood to be a generalization and advancement of edge computing [73].

IoT

The underlying development of increased embedding of computing in the physical world is the IoT, which is seen as an intersection of three paradigms, namely internet-oriented (middleware), things-oriented (sensors) and semantic-oriented (knowledge) [113]. Gubbi et al. define the IoT as an "*interconnection of sensing and actuating devices providing the ability to share information across platforms through a unified framework, developing a common operating picture for enabling innovative applications. This is achieved by seamless ubiquitous sensing, data analytics and information representation with Cloud computing as the unifying framework*" [113]. In other words, paradigms such as cloud computing are seen as enabling technologies for the IoT.

Blockchains

A very recent development in distributed systems is found in *blockchains*, also known as *decentralized consensus* [105], *decentralized computing* [83], or *decentralized applications* [217]. Blockchains can be understood as another strategy to avoid centralization, and as such are an orthogonal—rather than competing—technology to edge computing and the IoT. In fact, IoT applications can benefit from blockchain technologies, just like they benefit from cloud or fog computing. The core idea behind blockchains is that instead of performing a computational task on one system under the authority of one party (person or organization), it is performed by a massively distributed system using smart contracts [83], where not one authority must be trusted, but the required trust is distributed among all participants of this system [8]. This removes the necessity of trusting any single node, while maintaining trust into the technology as a whole [246]. This is enabled by using cryptographic mechanisms to ensure that the majority of the participants decides on the outcome of the computation, and a single attacker cannot manipulate the computation or its result. An attacker

must be able to control a substantial amount of participating computers to manipulate the overall system's behavior [85]. In contrast, a user can trust the overall system as long as the majority of participants are believed to be non-malicious.

Intrinsic and Extrinsic Metrics

Today, distributed systems maintain elasticity by monitoring certain *metrics*, analyzing the information, planning an action, and executing it [155], a loop commonly called MAPE. In this process, the underlying metrics play a crucial role, representing measurable attributes of a distributed system, and changing over time. Based on the value of one or multiple metrics, a distributed system can adapt to changes in the state of itself or its environment [41]. This can be achieved by scaling, or by using advanced adaptation mechanisms spanning multiple layers [200].

Various kinds of metrics exist. We mainly distinguish between *intrinsic* and *extrinsic* metrics. Intrinsic metrics are values measurable by inspecting the system state. Taking into account a resource executing a task, such as a VM, examples for intrinsic metrics include its CPU load [114], memory utilization [68], network traffic [274], or its overall performance [45]. Other intrinsic metrics include the memory utilization, the amount of Input/Output (I/O) activity, or the utilization of storage or network.

In contrast, extrinsic metrics are values measurable by inspecting the system's environment or context [99], such as the amount of requests (called *load* or *demand*) [129], or their attributes, such as cost and QoS goals [269].

2.2 Elasticity Approaches

Distributed systems can utilize metrics in various ways to maintain elasticity. A very simple—and yet widely used [178]—approach is using thresholds of metric values, and defining actions to be taken when thresholds are reached or exceeded. One example of such an approach could be the following set of instructions:

- *If the CPU load of any VM in the set of VMs exceeds 90%, the system is under high load. Activate one more VM.*
- *If the CPU load of all VMs in the set of VMs is below 10%, the system is under low load. Passivate one VM.*

Rule-Based Approaches

This example encompasses two concepts known in contemporary literature. First, *rules* are used to define the system behavior. Such an approach is called *rule-based*. Second, *thresholds* are defined for these rules, resulting in a *threshold-based* approach. This combination is very common [178], as it eventually maintains the system state desired by the operator (the person defining the rules) within

the defined thresholds. However, while the system state is kept between the thresholds in the long run, on a short time scale, the system may significantly violate the thresholds before its reaction shows effect. For instance, a sudden increase of load may need not one, but a number of additional VMs for the system to compensate. However, due to the way the rule-based approach is formulated, only one VM is activated at a time.

Naturally, one might counter this problem by defining additional rules, such as the following:

- *If the CPU load of any VM in the set of VMs exceeds 95%, the system is under* very *high load. Activate three more VMs.*
- *If the CPU load of any VM in the set of VMs exceeds 90%, the system is under high load. Activate one more VM.*
- *If the CPU load of all VMs in the set of VMs is below 10%, the system is under low load. Passivate one VM.*
- *If the CPU load of all VMs in the set of VMs is below 5%, the system is under* very *low load. Passivate two VMs.*

However, doing so poses additional issues. First, the thresholds must be chosen carefully, and assumptions must be made about the behavior of CPU load within the system. These assumptions are tightly coupled to the type of task executed by the system (e.g., whether it is heavy on CPU load, heavy on memory consumption, or heavy on I/O processing). Second, the rules must be created by an expert, and they must be maintained over time to adapt for concept drift [264], such as changing demand, different behavior of the VMs (e.g., a different type of CPU causing less severe spikes in CPU load), or different type of task executed. This can only be overcome by periodic reviews, which in turn require expert knowledge and time. Finally, ensuring exhaustion of all possible rules is difficult. If the expert creating these rules is unaware of a certain relationship between a parameter and the outcome, the missing rule goes unnoticed.

Concept Drift

The type of elasticity approach discussed above—that is, a system which is *reacting* to changes, for instance by following a set of rules—is called a *reactive approach* [178]. In contrast, we use a different class of approaches. Instead of merely reacting to changes in the system or its environment *a posteriori*, we are interested in creating systems that anticipate (likely or possible) changes, and act *proactively*. We call approaches using such techniques *proactive approaches* [178]. The main benefit of proactive approaches is that, if the system anticipates the change properly, there is no delay included before the system adapts to a change.

Reactive, Proactive, and Predictive Approaches

Naturally, the main challenge is accurately anticipating such changes requiring action by the system.

A specific subset of proactive approaches is a set of techniques called *predictive approaches* [178]. Predictive approaches not only anticipate a possible change and act proactively, but do so by projecting a concrete prediction about the development of metrics in the (near) future. In order to illustrate the difference between proactive and reactive approaches, we consider using a scaling approach in a cloud application using VMs. Maintaining a pool of hot-standby VMs (possibly from a cheaper source of computational resources, such as preemptive VMs), is a proactive elasticity approach: In order to ensure elasticity in anticipation of a load increase, hot-standby VMs are used at all times. However, no concrete prediction is created. In contrast, a predictive approach would, for instance, create a forecast based on the time of day, where a higher load is expected during day time, and the number of hot-standby VMs is increased, but during night time, VMs are spun down, since no demand peak is expected.

Note that reactive, proactive and predictive approaches are not mutually exclusive. In the illustrated example, a suitable approach would be to use predictions to determine the number of hot-standby VMs, but the transition from hot-standby to active would still be done reactively. Therefore, a system's behavior can be denoted as purely reactive, as proactive but featuring reactive elements, or as predictive, which includes proactivity and also reacts to changes not anticipated by the prediction.

2.3 Prediction Techniques

Classification, Regression, Clustering

Various methods exist for predicting future values of metrics, ranging from relatively simple approaches like linear regression to techniques like ANNs [119], Support Vector Machines (SVMs) [147], or Genetic Algorithms (GAs) [110]. Any of these techniques can be used for ML, and the result of their usage is called the *ML model*. Three main types of tasks performed by ML are classification, regression, and clustering [138]. Classification is the process of assigning a tuple of data to one of a finite number of discrete classes. For instance, the tuple

(*Skin:* Furry, *Legs:* 4, *Whiskers:* No, *Tail:* Yes, *Height:* 60cm) → *Animal:* ?

may be classified by a ML classification model as *Animal:* Dog. In contrast, regression is the process of assigning a tuple of data to a continuous variable [216]. For instance, the tuple

(*Skin:* Furry, *Legs:* 4, *Whiskers:* No, *Tail:* Yes, *Height:* 60cm) → *Weight:* ?

may be classified by a ML regression model as *Weight:* 18kg. Note that while in the classification example, the type (class) of animal was sought, in this

example, we are interested in the weight, which, in contrast to the type, is continuous. There are infinitely many values the animal's weight can be (even in a bounded context), while there is a finite number of animal types. Finally, clustering is the process of grouping together tuples belonging into the same set (sub-populations) [234].

In regression, classification, and clustering, the elements given to the ML model as input data are called *features*. For regression and classification, the element expected as output is called *label* [175]. Clustering does not operate on labels, but tries to find groups of similar input data (clusters).

Supervised, Unsupervised and Reinforcement Learning

The main benefit of ML is that in contrast to rule-based systems, which can also be used for classification or regression, the model is *trained* on data, and once trained, can be used for classification or regression. There are three fundamental approaches to training ML models [228]: If the training data consists of a dataset including labels (known correct outputs), we refer to the process as *supervised learning*, since the model can be supervised and feedback about the correct output can be provided. In contrast, if the training data only contains features, but no labels, the process is called *unsupervised learning*. Since no labels are present for unsupervised learning, the ML model can only react to common patterns in data, as well as the presence and absence of certain items in the training data. Finally, *reinforcement learning* [178] is the process of training a ML model to maximize a certain reward by performing actions. While reinforcement learning is related to supervised learning (e.g., feedback is provided in both scenarios), it differs in the fact that no notion of *correct/incorrect* is required, and as such, no explicit labels are required. Instead, values, possibly taken from the real world, can be presented to the ML model as feedback [56]. One such example would be a ML model learning to fly an airplane by controlling the input on the three main axes, with the goal of reaching a given altitude. The reward for the ML model is higher if the airplane is flying closer to its assigned altitude, and the ML model would try and maintain the maximum reward.

Online and Offline Learning

Another distinction in ML is the temporal classification of training techniques [18]. The approach of supplying the ML model with a set of data for training, and then using the trained ML model in operation, is called *offline learning*. In contrast, *online learning* is used in situations where data is made available over time (one by one). Outputs (classes for classification, values for regression, clusters for clustering) are required during operation, but the ML model is also trained with each set of data fed during operation, one at a time [167]. This is required especially for predicting live data, such as stock market prices.

Precision and Recall

For classification, *precision* and *recall* are important performance metrics for a ML model. Precision and recall are based on the four commonly used confusion matrix metrics True Positive (TP), False Positive (FP), True Negative (TN), and False Negative (FN) [182]. With respect to a given class C, TP denotes

instances correctly classified as C, TN denotes instances correctly classified as $\neg C$ (not C). Correspondingly, FP denotes instances incorrectly classified as C, and FN denotes instances incorrectly classified as $\neg C$. Based on these values, the definitions of precision and recall are shown in (2.1) and (2.2), respectively.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.1}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2.2}$$

Precision determines the fraction of correctly-classified instances of C, relative to all instances classified as C ("Out of all classifications of C, how many instances actually belong to C?"). Recall determines the fraction of correctly-classified instances of C, relative to all *actual* instances of C ("Out of all actual instances of C, how many instances are classified as C?").

MCC

Precision and recall show different qualities of a classification model. In cases where a single metric is required, we use the Matthew's Correlation Coefficient (MCC) [188], as defined in (2.3). The MCC unites all four confusion matrix metrics. All three, precision, recall, and MCC, are commonly used metrics for evaluating binary classification algorithms [188, 213, 240]. All three values are metrics for the goodness of fit of a ML model, i.e., higher numbers denote better performance. Precision and recall are bounded to $[0; 1]$, while MCC is bounded to $[-1; 1]$. Negative MCC values denote performance worse than random selection.

$$\text{MCC} = \frac{\text{TP} \times \text{TN} - \text{FP} \times \text{FN}}{\sqrt{(\text{TP} + \text{FP})(\text{TP} + \text{FN})(\text{TN} + \text{FP})(\text{TN} + \text{FN})}} \tag{2.3}$$

RMSD

For regression, there is no binary distinction between correct and incorrect classification (TP, TN, etc.). Therefore, we use numeric values to denote a ML model's performance. One commonly used metric for evaluating the quality of regression is the Root-Mean-Square Deviation (RMSD) [141]. For a regression model M and a set of tuples T, the RMSD is defined as shown in (2.4), where $L(t)$ is the actual label of tuple t, and $M(t)$ is the output of the ML model, i.e., the regression value.

$$\text{RMSD}(M) = \sqrt{\frac{\sum_{t \in T}(L(t) - M(t))^2}{|T|}} \tag{2.4}$$

2.4 Artificial Neural Networks

In this work, we make heavy use of ANNs [119] as the ML model of preference. While choosing a ML model, we take into consideration the fair amount of research in the various areas. ANNs have been shown to provide improved performance over other ML models [181]. Compared to models like SVMs [147], Bayesian classification or Fisher's linear discriminant [26], feed-forward ANN models with error backpropagation are well-suited for regression and provide efficient means of statistical pattern recognition while allowing to use compact models with sufficient generalization performance [26].

Artificial Neurons

ANNs are based on individual units called *artificial neurons*, modeled after biological neurons [26]. A neuron takes a finite and constant number m of continuous values as inputs (x_1 to x_m), and produces one continuous value as output (y). This is achieved by multiplying each input x_j (where $1 \leq j \leq m$) with a weight w_j, and adding all resulting weighted inputs. In addition to the weighted inputs, an additional input x_0 with the constant value of $x_0 = 1$ is added, together with its own weight w_0. This weight w_0 is called the bias. The so-called *activation function*, denoted as ϕ, is then applied to the sum of all weighted inputs (including the bias). Traditionally, both the inputs and outputs of artificial neurons are normalized to a range such as $[0; 1]$ or $[-1; 1]$ [214], or a distribution with mean $\mu = 0$ and standard deviation $\sigma = 1$ [25]. Such normalization has been shown to greatly improve prediction performance and to reduce training time [241].

Activation Function

The formula for the output of a single neuron is shown in (2.5).

$$y = \phi\Big(\sum_{j=0}^{m} w_j x_j\Big) \tag{2.5}$$

An ANN is defined as a network of neurons. There are various classes of ANNs, and for demonstration purposes, we describe the most common type. We will discuss special types of ANNs in more detail in Chapter 4, where various refinements are used.

Neuron Layers

The neurons of an ANN are aligned in *layers* [119]. There is at least one layer of neurons (the *output layer*), and traditionally, a dedicated *input layer* is also used. Additionally, a number of layers between the input and the output layer is used, called *hidden layers*. The inputs of each neuron in each layer are the outputs of all neurons of the previous layer. In other words, if one layer has a neurons, each neuron in the next layer has a inputs. If the next layer has b neurons, there are $a \times b$ input-output connections. Figure 2.1 shows an example network consisting of an input layer with two neurons, one hidden layer with ten neurons, and an output layer with one neuron. The amount of hidden

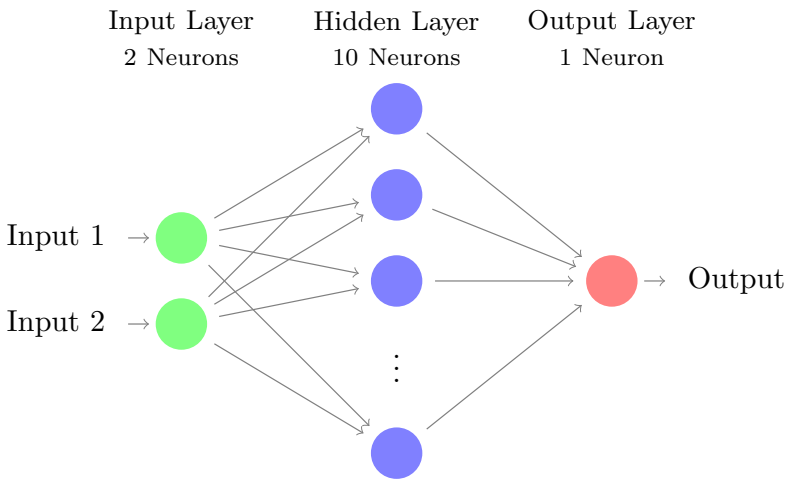

Figure 2.1: Example of a multi-layer ANN

layers, as well as their respective number of neurons, vary between ANN models, depending on their use case.

The inputs of the input layer are the input of the ANN itself (i.e., the features), and the outputs of the output layer are used as the overall output of the ANN. The output can be treated as the ANN's "best guess" for the label corresponding to the features presented at the input layer. In the context of (future) value prediction, the ANN output represents the network's prediction.

The input (and bias) weights of each neuron within an ANN determine the ANNs behavior. Therefore, finding suitable values for the weights is the core goal of training. Without training, a human would have to find suitable weights for each neuron, which—as mentioned in the downsides of rule-based elasticity approaches explained in Section 2.2—we aim to avoid.

Backpropagation

While research of the concepts behind ANNs, called Logical Calculus [190], dates back to 1943, a key contributing factor for the uprise in the last decades was the backpropagation algorithm [120, 222]. Together with the advent of massive computation power available for training, such as using parallel distributed computing or performing computations using a General-Purpose Graphics Processing Unit (GPGPU) [207], ANNs became a feasible ML model, and a commonly used tool for ML tasks.

Training using backpropagation works by presenting one tuple of training data (features) to the input layer, and reading the output provided by the network. The difference between the actual output and the correct label (the error) is then propagated into the network, adapting the network's weights, so that the next time this particular tuple is presented to the ANN, the resulting output will be closer to the correct label. The rate with which the weights are

adapted is called *learning rate*. A lower learning rate requires more training iterations (and therefore, time) to train the ANN. On the other hand, a higher learning rate increases the risk of the network over-compensating for individual tuples and not converging to a usable ML model.

2.5 State of the Art

As discussed in Chapter 1, resource provisioning is crucial for implementing elasticity in distributed systems [60]. In Section 2.1, we have discussed how various types of metrics can be used to realize such elasticity, and shown described types of approaches to elasticity. In our discussion, we have seen how proactive approaches, as opposed to reactive approaches such as rule-based systems, can provide elasticity without increasing the demand for expert knowledge. In this section, we provide a high-level overview over the state of the art and its gaps with respect to resource provisioning.

In the following, we classify existing resource provisioning approaches by the question answered in the respective approach. Using this classification, we identify three core resource provisioning questions, namely scaling, placement, and scheduling. These aspects have been often discussed in the context of cloud computing [9, 27], but are crucial also in other fields of distributed systems, such as DSP [63] and elastic BPM [232].

Scaling The ability to dynamically scale in and scale out is a fundamental feature of contemporary elastic distributed systems. As highlighted in Section 2.1, the scaling decision is a non-trivial problem [62]. Often, there are multiple, conflicting optimization target metrics. For instance, a service provider is interested in reducing the total cost of operation of the infrastructure, but at the same time, wishes to provide a given QoS, or needs to fulfill certain SLAs in order to avoid penalties. In some situations, such penalties can be expressed as a cost factor, which reduces the number of target variables to be optimized. However, in other situations, the impact of SLA violations exceeds the impact of additional cost incurred by penalties. For instance, negative reputation generated by substandard QoS might be hard to express as numbers [5]. In such cases, additional non-cost constraints must be introduced.

Placement Task placement is the problem of assigning resources to tasks. Like scaling, it is one of the most researched problems in various types of distributed systems, including cloud [27, 118] and DSP systems [52]. As discussed in Chapter 1, the problem of placement can occur between different layers of a system, such as the placement of applications on VMs, or VMs on PMs [177].

Placing tasks on resources can be optimized in various ways. The placement of tasks occupies workers for a certain amount of time, and therefore, the exact constellation of tasks causes significant changes in the total cost of operations. Furthermore, various tasks can be co-located (consolidated) on the same worker in order to save resources, or to reduce latency by avoiding unnecessary network transmissions [62]. On the other hand, placing certain tasks on the same worker might have negative impact, e.g., when a high concurrency of workers regarding a certain resource leads to a situation of competition, where the distribution of a limited resource among the workers is uneven, or, in the extreme case, *resource starvation* occurs [153]. In such cases, dedicated distribution of applications is required [163].

Scheduling In contrast to the placement problem, which deals with the question of *where* to place tasks, the scheduling problem arises when asking *when* to run a given task. Placement and scheduling are related problems and sometimes addressed as one [198, 259], aiming at similar goals, including optimized resource usage [153], adaptive (self-tuning) features [118] and network efficiency [62].

A holistic overview of resource provisioning strategies and approaches is provided by De Assunção et al. [11]. In this survey, the authors describe an extensive classification taxonomy of approaches and techniques. Many of these approaches select a specific metric and aim for optimizing operations with respect to this metric: Meng et al. [194] discuss traffic-aware VM placement. The goal of this approach is to reduce the aggregate traffic flowing into a data center. The authors formulate an optimization problem and design an approximate algorithm, based on clustering. They use traffic traces collected from production data centers to evaluate their approach. The approach presented by the authors is applicable to various classes of distributed systems, not limited to cloud computing. In contrast, Cardellini et al. [52] discuss a placement approach for DSP applications based on an Integer Linear Programming problem, explicitly taking into account the heterogeneity of a system's resources. Beloglazov et al. [17] apply a best-fit algorithm enriched by the aspect of energy efficiency. The effect is the consolidation of VMs on as many physical hosts as possible, in order to minimize total cost of operation and energy consumption, leading to a reduced carbon dioxide footprint. Additional solutions include analyzing the network topology, as shown by Biran et al. [24], where the Min Cut Ratio-Aware VM Placement Problem is described, and an approach to VM placement while reducing the worst-case load ratio over topology cuts is shown.

Work by Lorido-Botran et al. [178] provides an overview of proactive approaches in cloud environments, with a focus on elastic applications. The study compares rule-based and proactive approaches and highlights the potential of prediction-based solutions. Furthermore, the authors advocate the necessity for real-world

data traces for evaluation purposes. While the study is focused on cloud environments, many of the approaches referenced by the authors are not limited to this type of distributed system.

Prominent predictive solutions include work by Napoli et al. [204], where wavelet analysis (a time series analysis methodology) and recurrent ANNs are used to perform load prediction, and Islam et al. [140], where the authors present a resource measurement and provisioning strategy, again using ANNs. Caron et al. [58] also perform workload prediction using time series analysis by providing an auto-scaling algorithm.

As a special case, Gandhi et al. [101] perform resource provisioning using a hybrid approach with both predictive and reactive elements. A predictive method is used for coarse, long-term time scales (e.g., hours), i.e., the distribution of the base load is being predicted for these scales. Additionally, the reactive provisioning component handles fine-grained, short-term peaks not predicted by the long-term predictor component.

Nevertheless, many aspects of predictive resource provisioning remain unexplored. First, on a higher-level view, an analysis of the applicability of certain methods in certain situations would greatly benefit system designers and architects in the domain of distributed systems when selecting an approach or technique for a given use case. While individual literature describes individual methods on a per-approach basis, holistic overviews, apart from the studies mentioned above, remain scarce. The remainder of this thesis aims to address this gap not only by providing additional empirical evidence for applicability of certain predictive methods in example scenarios, but also by putting these methods in context with each other, and therefore gaining additional systematic knowledge about the application of predictive methods in distributed systems. Second, on the level of the three resource provisioning problems described above, many detailed aspects have also not yet been extensively studied. For instance, the per-task estimation of resource utilization for placement and scheduling, or the fusion of intrinsic and extrinsic sources using EKF-based filtering or ANNs, remain unexplored. The following chapters of this thesis aim to provide detailed discussions and in-depth evaluation of these gaps.

CHAPTER 3

Predicting Resource Utilization

As highlighted in Section 2.1, elasticity is an important property of contemporary distributed systems. In this chapter, we investigate the utilization of resources in a cloud computing system, such as CPU time, memory, and storage, since understanding the system's resource utilization and being able to make predictions about its future development are core requirements for maintaining elasticity of a distributed system in a predictive manner.

We propose the following example scenario as motivation and guide for the work presented in this chapter.

> **Example Scenario.** The provider of a cloud-based service for building, testing, integration, and deployment of software systems, referred to as Continuous Integration (CI), is serving thousands of clients daily. Software development companies use this cloud-based CI service for their development process. Whenever a developer commits code to a repository, this code is submitted to the cloud-based CI service offered by the provider, built, tested, and packaged. The CI provider reports on the outcome of the individual steps (e.g., whether the build was successful, which tests failed, etc.), and, if no failures occur, deploys the software to testing and production environments. The provider hosts the CI service on a cloud-based infrastructure, therefore benefiting from potential scalability and elasticity. To properly maintain this elasticity, the provider uses predictive scaling approaches. Instead of waiting for an increase in demand and reacting by scaling out the service, the infrastructure predicts the anticipated increases, which improves the QoS experienced by the clients.

3.1 Fundamentals

As stated in Chapter 1, resource provisioning is one of the main tasks for a cloud infrastructure. Therefore, a central software component is typically dedicated to this task. In line with related work [145], we refer to this component as the *provisioning agent*. The provisioning agent is responsible for the placement of tasks (i.e., assigning a computational resource such as a VM to a given task), their scheduling (i.e., deciding when a given task should be executed), and the system scaling (i.e., deciding whether additional resources are required, and to what degree the system should scale out or scale in).

Resource provisioning and resource management are vivid fields of research in cloud computing, resulting in a large number of solutions for reaching placement, scheduling, and scaling decisions [273]. Often, the focus of resource management is put on QoS-aware provisioning under given cost constraints, or under another set of rules [238]. Apart from a large number of general solutions for the SaaS [268], PaaS [4], and IaaS [171] paradigms, more specific solutions exist, e.g., for the execution of scientific workflows [247], business processes [134], or DSP [131] in the cloud.

While the individual aims of cloud resource provisioning solutions differ substantially, e.g., taking into account QoS and SLAs [238], or optimizing the execution of complex business processes in the cloud [134], the underlying approach is similar throughout literature: The goal is to distribute task requests onto cloud-based computational resources such as VMs or containers [136]. In this chapter, we consider a cloud computing platform which operates using VMs. Incoming tasks are assigned to a VM by the provisioning agent, depending on the nature of the task, the expected resource utilization, as well as the current system state. If the system load is already high and more resources are required, additional VMs are spun up. Likewise, if the system load is too low, tasks are consolidated, and VMs are shut down in order to save cost. This behavior, as explained in Section 2.2, is typical for contemporary cloud computing systems.

For maintaining system scalability in the cloud using predictive approaches, it is crucial to forecast the amount of resources its incoming tasks will utilize when executed. Based on this data, the provisioning agent is able to accurately perform its tasks, i.e., scheduling, placement, and scaling. Due to its proactivity, the provisioning agent can perform these tasks in advance, instead of merely reacting to the observed level of resource utilization [140].

To address the challenge of efficient resource provisioning, we present a generic approach to predicting cloud resource utilization. We show how to use traces of recorded executions of tasks submitted to a cloud-based service, along with their resource utilization, to build a ML model capable of predicting the resource utilization of tasks submitted in the future. In our scenario, we process labeled data, i.e., task submissions, and the resulting resource utilization values, to

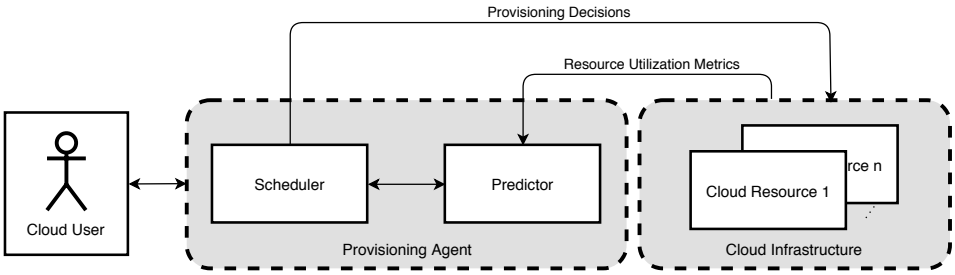

Figure 3.1: Proposed cloud platform predicting resource utilization

create a regression model for subsequent tasks. As the underlying rules and patterns are not known, we use ANNs, since as described in Section 2.4, they are well-suited for regression and pattern recognition while maintaining sufficient generalization performance [26]. In the evaluation of this chapter, we apply this approach to predict the duration (run time) and CPU utilization of tasks submitted to a CI service.

3.2 Prediction of Resource Utilization

For predicting resource utilization in the cloud, we propose the employment of techniques from the field of ML. The intuition behind this approach is that the resource utilization of a cloud-based task might not scale linearly with the cardinality of its input. Instead, non-linear behavior is possible, rendering simple linear regression insufficient. Furthermore, some types of tasks submitted to a cloud-based service—such as the CI service described in the example scenario—might take a vector of input data instead of a single item, which would require multivariate linear regression, substantially increasing the parameter space. Therefore, we use ML in order to create prediction models from historical data, i.e., data stemming from past task executions, and use this model for obtaining future predictions.

3.2.1 Proposed Architecture

In Figure 3.1, we show the overall architecture of our proposed approach. A *client*, i.e., a consumer of a cloud-based service, posts a request for the execution of a task to the system, which is governed by the provisioning agent. The provisioning agent itself is composed of two components: the *scheduler*, which is responsible for performing task scheduling and provisioning, and the *predictor*, which supports the scheduler by predicting resource utilization. This enables the scheduler to optimize its decisions. Applying our example scenario, the CI provider hosts both the provisioning agent and the cloud infrastructure, e.g., VMs, on which the CI tasks submitted by the development companies (represented by the cloud user in Figure 3.1) are placed and executed.

Relation to Example Scenario

Upon receiving a task execution request from the client, the provisioning agent must decide on how to provision the cloud resources. In contemporary approaches, this involves finding a schedule (time) and selecting concrete computational resources (placement) for the submitted task [133]. Such a decision requires knowledge about the assumed duration and resource consumption of the task. For this, the scheduler queries the predictor to receive a prediction of those metrics. Based on the decision reached by the provisioning agent, the task can then be scheduled and placed, and eventually executed by the infrastructure.

After the execution of the task, the cloud infrastructure reports the measured (actual) resource utilization back to the predictor. Recorded traces of this data, i.e., a history of predicted and actual resource utilization values, are used by the predictor component to train its model.

In this chapter, we investigate in detail how the predictor component can use the history of recorded traces of tasks executed on the cloud infrastructure, i.e., the type of task and its attributes, to achieve accurate utilization prediction for individual resources.

3.2.2 Machine Learning Model

Features

In order to formalize our approach, we define the problem of predicting the resource utilization for a given task to be provisioned. We assume that the provisioning agent is supplied with the following information (features) about the task (e.g., by the client submitting the task):

- The type $\mathcal{T}$ of task to be provisioned
- A vector $(a, b, c, \dots)$ of input data for the task

Labels

Furthermore, a list of resources for which utilization levels are to be predicted (labels) is known a priori. These resources are denoted as $R_1, R_2, \dots, R_n$ and can represent CPU time, execution duration, the utilization of memory or storage, or any other form of resource necessary to execute a task. Thus, the variables $R_1, R_2, \dots, R_n$ can be of any type, e.g., Boolean values, integers, or decimals, as long as they can be encoded in real numbers, which constitute the input domain for ANN models.

For each resource R_i, a prediction of utilization $\widehat{R_i}$ is returned by the ML predictor. Therefore, the result of a prediction consists of a single data item:

- A vector $(\widehat{R_1}, \widehat{R_2}, \dots, \widehat{R_n})$ of resource utilization predictions

To solve this problem of resource utilization prediction, we use ANNs as described in Section 2.4. To this end, a prediction model $\widehat{\mathcal{T}}$ is created for the resource

utilization of every task type $\mathcal{T}$ that requires such a prediction. The predicted resource utilization subsequently allows for an optimized provisioning.

Each execution of a task of type $\mathcal{T}$ with an input vector $(a, b, c, \dots)$ produces a vector of measured (actual) resource utilizations $(R_1, R_2, \dots, R_n)$. This resource utilization vector is recorded. After collecting a sufficient amount of records, the data is used to train the ML model. With an increasing amount of training data, the ML model should become more and more fit, and therefore its predictions should show increasing accuracy. We have discussed in Section 2.3 that such ML training approaches can be classified into online and offline approaches, i.e., the training can happen during the execution or beforehand. In our current approach, we employ offline learning by using collected data traces to train our ML model, however, applying online learning is also feasible without substantially increasing complexity. In Chapter 4, we show how ANNs can be used with online learning.

Once trained, the ML model is able to predict the resource utilization vector to a certain degree. In other words, the model can then be presented with a task type $\mathcal{T}$ and an input vector $(a, b, c, \dots)$, and produces a resulting prediction vector $(\widehat{R_1}, \widehat{R_2}, \dots, \widehat{R_n})$.

ANN Topology

While conducting research in order to obtain a set of parameters for the ANN we use as a ML model, we found that using an ANN with a single hidden layer is suitable for most scenarios, including curve fitting, which our use case represents [220]. We furthermore found that using Stochastic Gradient Descent (SGD) [40] together with Nesterov's Accelerated Gradient (NAG) [205, 206] is an established weight update method [245]. The weights are initially filled by the means of the *Xavier* algorithm, based on work shown in [106], which initializes each neuron's weights using $X \sim \mathcal{N}_{\mu,\sigma}$ with $\mu = 0$ and $\sigma = \frac{1}{n_{\text{out}}+n_{\text{in}}}$, where n_{out} and n_{in} are the numbers of output and input connections, respectively, of the neuron. Table 3.1 summarizes the configuration of our ANN model.

In the ANN used in our approach, the input vector is presented to the input layer neurons, which produce an output according to their weights. This output is passed to the hidden layer. The neurons in the hidden layer process their input values according to their weights, and their output is passed on to the output layer neurons (one for each output variable). The output generated by those neurons is then considered the output of the ANN, and therefore, the output of the ML predictor. Figure 2.1 provides a graphical illustration.

Training and Validation

In the *training* phase, i.e., after the execution, the predicted resource utilization value obtained from the network output is compared to the actual (measured) resource utilization. The error is used to update the neuron weights according to the aforementioned NAG method. In the *validation* phase, i.e., for evaluation, the value obtained from the network output is also compared to the actual resource utilization, but the network is not updated, since it is already assumed to be trained. As discussed in Section 2.3, this is in contrast to online learning,

Table 3.1: ML model parameters used in this chapter

Parameter	Value
Input layer neurons	2
Hidden layers	1
Hidden layer neurons	10 per output
Output layer neurons	2
Activation function	tanh
Learning rate	0.01
Minimization algorithm	SGD [40]
Weight initialization	Xavier [106]
Weight update	NAG [205, 206]
Weight update momentum	0.9
Training epochs	250

where training is a continuous process. The values obtained from the network output, as compared to the actual utilization measurements, are used to evaluate the network's performance according to the metrics described in Section 3.3.1.

3.3 Evaluation

We thoroughly evaluate the performance of the proposed ML predictor using various experiments. In our evaluation, we aim to predict the CPU utilization and duration of cloud-based task executions as two exemplary target variables. In other words, we regard the CPU utilization and time required by a task as resources (R_1 and R_2, respectively), and create according prediction functions ($\widehat{R_1}$ and $\widehat{R_2}$, respectively) as defined in Section 3.2.

3.3.1 Baseline and Metrics

In order to determine the degree to which ML improves the prediction of resource utilization, we require a suitable baseline. In contemporary literature, various approaches to predicting this utilization exist, as we discuss in Section 3.5. To establish a proper common ground for performance comparison, we have identified linear regression as a common element of these approaches. We therefore use both a basic linear regression predictor—the baseline—and our proposed ML predictor in our experiments.

It is necessary to observe differences between the predictions $\widehat{R_1}$ and $\widehat{R_2}$ generated by the predictor (baseline or ML), and the actual values R_1 and R_2. For this, we use the RMSD, defined in (2.4). Note that when referring to the RMSD, we either explicitly denote a specific variable for which the RMSD is calculated (e.g., the RMSD of R_1), or use the RMSD without regard to a specific variable, in the

context of an aggregation. In the latter case, we aggregate the RMSD values for both R_1 and R_2 using their arithmetic mean.

For a given target variable R_i, we designate the RMSD of the baseline as RMSD_B, and the RMSD of our ML approach as RMSD_ML. Following this, we consider the impact of our approach, and define its *error ratio* for a given resource R_i in a given experiment as:

$$\text{error ratio}(R_i) = \frac{\text{RMSD}_\text{ML}(R_i)}{\text{RMSD}_\text{B}(R_i)} \tag{3.1}$$

We use this metric to be able to compare deviations across metrics regardless of their order of magnitude and unit. An error ratio of 1.0 indicates that in a particular experiment, the ML approach achieves the same result as the baseline approach. Any error ratio above 1.0 indicates that the ML approach performs worse than the baseline, and a value below 1.0 indicates that the ML approach yields a lower RMSD than the baseline, thus performing better.

3.3.2 Evaluation Dataset

To evaluate the proposed prediction approach, we created an extensive dataset, using Travis CI[1] and GitHub[2] as data sources. Travis CI is a publicly available CI service for projects hosted on GitHub. For instance, when a developer pushes new code to GitHub, the CI server detects this change, and triggers a new build–test–package cycle. Travis CI provides information about this CI process openly and without restriction unless a project is using a premium plan, in which case the results of the CI process can be hidden. Such results are therefore not part of our dataset. This data includes—among others—the name of the project (repository) that has been built, tested, and packaged, the commit ID, and the build duration.

Data Sourcing

We gathered the evaluation dataset by creating a crawler program based on the Travis CI Application Programming Interface (API) to collect data about project builds. We call the records in this dataset *build process records*. The crawler was executed for a total time of roughly one week. We enriched the data with the GitHub commits' file counts and total sizes in bytes. By using the aforementioned API, we collected a raw dataset of over 3 million Travis CI build process records from over 35,000 GitHub repositories. Out of these, we removed repositories no longer publicly available (e.g., due to deletion or restriction). We furthermore excluded repositories larger than 200 MB, since our analysis later requires downloading code, which was not practical without a given size threshold. For each repository, we recorded the main programming language

Pre-Processing

[1] `https://travis-ci.org/`
[2] `https://github.com/`

Table 3.2: Ten most frequent evaluation languages in the dataset

Language	Build records	% of total
JavaScript	241,328	21.2%
Python	192,363	16.9%
Ruby	157,280	13.8%
Java	119,214	10.5%
PHP	103,535	9.1%
C++	71,359	6.3%
C	58,350	5.1%
Go	28,972	2.5%
Scala	21,422	1.9%
Objective-C	20,169	1.8%
(other)	*124,441*	*10.9%*
Total	1,138,433	100.0%

used, to test the hypothesis that the language indicates the technology used, and impacts the resource utilization of the build task. Information about the main programming language is provided natively by GitHub.

Finally, since we were lacking CPU utilization information from Travis CI, we added this metric to the dataset by randomly picking build process records which were marked by Travis CI as successful, and running them on our private cloud infrastructure. We recorded the duration and average CPU utilization of these runs, and correlated this information with the data collected from Travis CI. From this, we projected the CPU utilization to the rest of the dataset. We only consider processes where our build succeeded, i.e., unsatisfied dependencies were ignored by dropping the build process record from the dataset.

Summarizing, the effective dataset we use for training of our resource utilization prediction approach consists of tuples with the structure shown in (3.2), where $\mathcal{T}$, a, and b are features, and R_1 and R_2 are labels.

$$\begin{aligned} < \mathcal{T} = (\text{Repository}, \text{Language}), a = \text{File Count}, b = \text{Size}, \\ R_1 = \text{Duration}, R_2 = \text{CPU Utilization} > \end{aligned} \tag{3.2}$$

The processed dataset consists of more than 1.1 million build process records from around 10,500 repositories. The distribution of the ten most frequent programming languages used inside this dataset is shown in Table 3.2.

For training, the data is stripped of its labels R_1 and R_2, and the predictions $\widehat{R_1}$ and $\widehat{R_2}$ are calculated using both predictors (baseline and ML). The errors

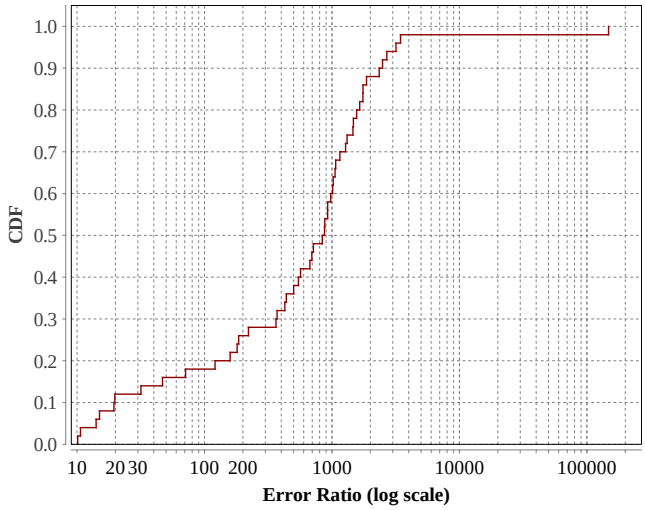

Figure 3.2: Weak performance of per-language ML models

between R_1 and $\widehat{R_1}$ as well as R_2 and $\widehat{R_2}$ are then provided to the model as training input for backpropagation. For evaluation, the predictions $\widehat{R_1}$ and $\widehat{R_2}$ are compared to the actual values R_1 and R_2, using the RMSD and error ratio metrics, as described in Section 3.3.1.

3.3.3 Experiment Methodology

Per-Language Performance

We conduct different preliminary experiments in order to assess the influence of various features on the prediction accuracy. In particular, we test whether the programming language alone provides a sufficient level of prediction performance. This way, repositories previously unknown to the predictor could be assessed using an existing per-language ML model, without an initial learning phase.

However, our analysis shows that aggregating builds per language, i.e., from many repositories, drastically reduces prediction accuracy. Figure 3.2 shows the Cumulative Distribution Function (CDF) of the error ratio in all languages. We see that the error ratio is in the orders of thousands. The median (CDF = 0.5) error ratio is around 800, indicating that the ML approach has an error 800 times as large as the baseline approach. We therefore conclude that per-language ML models are not feasible using our current ANN configuration.

Per-Repository Performance

As a next step, we split the dataset by repository. In other words, we train one ML model per repository, instead of training one per language. This also seems intuitive given the fact that our initial experiments have shown that individual repositories have very different build strategies. Generalizing from a CI task—used in our example scenario—to an abstract task in a cloud computing platform—the generic use case—this step corresponds to training one ML model

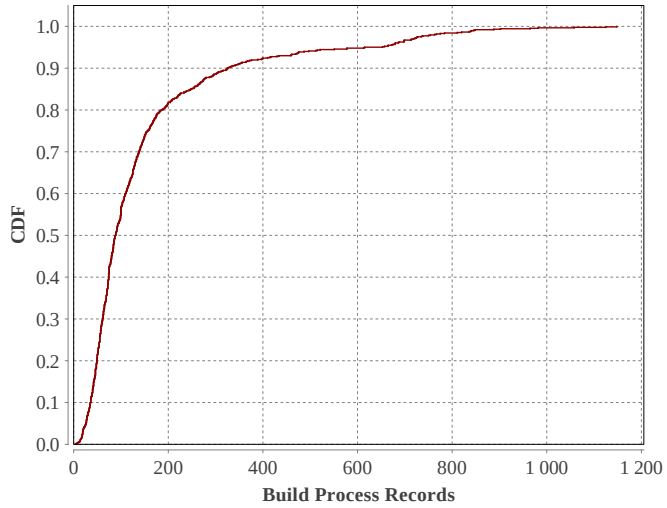

Figure 3.3: Distribution of build process records per repository

per scheduled task type $\mathcal{T}$ (e.g., a task executed by a certain service), instead of training one global model.

Therefore, we group the build process records by repositories. For this, we consider the amount of training data—that is, build process records—per repository, since this determines the training data available for a single ML model when using per-repository ML models. Figure 3.3 shows the distribution of numbers of build records per repository. On the horizontal axis, the number of build process records per repository is shown, and the vertical axis shows its CDF. A vast majority of repositories contains less than 200 build process records. In other words, for most of the repositories, the size of the dataset used for training and validation is less than 200. In the field of ML, this is a rather small sample size. Nevertheless, we will show in Section 3.4.1 that the performance is better than the baseline approach even for small training sets.

Training and Validation Sets

After having grouped the build process records, we randomly split them into *training* and *validation* sets, with a split ratio of 70% and 30%, respectively. We use these sets for training our ML models, as described in the following section.

3.3.4 Machine Learning Implementation

As discussed in the previous section, we use one ML model for each repository, not taking into account the language used. We therefore regard the repository as the task $\mathcal{T}$ requiring execution. Furthermore, we use two parameters of the build task: the amount of files in the repository at the time of the build, and the total repository size in bytes. Finally, the variables we seek to predict are the duration and CPU utilization of the task.

Table 3.3: ML model variables

	Variable	Class	Type	Example
$\mathcal{T}$	Repository	Nominal	Feature	`jlord/git-it`
a	File count	Numeric	Feature	113 files
b	Total size	Numeric	Feature	82 kB
R_1	Build duration	Numeric	Label	32.8 s
R_2	CPU utilization	Numeric	Label	82.8%

We therefore formulate the following problem with respect to Section 3.2.2: For each task, the input variables are the repository itself ($\mathcal{T}$), as well as the file count (a) and the total size (b) of the repository. The variables we are predicting are the task duration (R_1) and the CPU utilization (R_2). An overview of these variables is given in Table 3.3.

While ANN models generally accept the entire set of real numbers and thus the input domain of an ANN is not limited, ranges between 0 and 1, or numbers around 0 with a standard deviation of 1 are used due to the fact that activation functions used in most ANN models operate in this region.

We perform normalization as discussed in Section 2.4 by measuring the mean and standard deviation of the input training set, and then using linear normalization in the form of $\frac{x-\mu}{\sigma}$, in order to achieve a distribution with $\mu = 0$ and $\sigma = 1$.

For ANN implementation, we use the open-source Java library Deeplearning4J[3], which, among other features, provides built-in support for multi-layer ANN models and backpropagation.

3.4 Performance Analysis

In this section, we evaluate our ML prediction approach, based on the methodology presented in Section 3.3. We use the error ratio shown in (3.1) as an overall performance metric.

In Section 3.4.1, we analyze the impact of the amount of training data on the prediction performance in order to judge how well prediction works for repositories with few build records. In Section 3.4.2, we provide examples and an in-depth analysis of individual trained ML model instances. Finally, in Section 3.4.3, we provide an overall discussion of the results obtained.

[3] `http://deeplearning4j.org/`

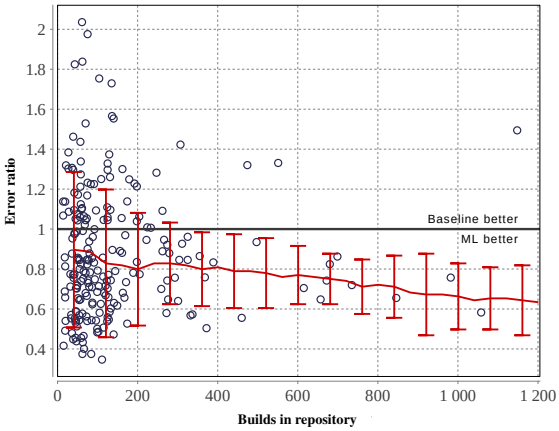

Figure 3.4: Error ratio over builds per repository

3.4.1 Impact of Training Data Amount

We argue in Section 3.3.3 that the sanitized dataset has to be split into groups per repository, since splitting per language yielded too diverse behavior and unsatisfactory results. However, this split drastically reduces the amount of data a prediction model receives for training. We recall Figure 3.3, which shows the distribution of available historical task executions (i.e., build process records) per repository. About half of the repositories in the dataset (CDF = 0.5) have no more than 100 build process records, and 95 % of all repositories have 300 build process records or less.

Therefore, our first goal is to assess the relationship between the amount of build process records and the error ratio. Figure 3.4 shows an overview of the repositories in the dataset, plotted to display their amount of build process records, and the resulting error ratio. We see that while the variance of the error ratio is clearly higher for low numbers of training data (low number of builds per repository), the increase of the mean error ratio for such repositories is minor (0.89 highest, compared to 0.85 total mean). Low amounts of executions for a certain repository yield a high variance, but on average, the performance is still better than the baseline (error ratio below 1.0).

To confirm this observation, we perform a control experiment. We strip the dataset of all repositories with the number of build process records lower than T. We use varying thresholds T for this data filtering. The result of one such filtering (using $T = 50$) is shown in Figure 3.5. The scatter plot shows that while the filtering is clearly visible, there is still a high amount of outliers, and the filtering does not significantly increase the performance deviation.

Note that Figures 3.4 and 3.5 refer to the same dataset. To maintain readability, stochastic subsets have been used for visualizing each dataset. Therefore,

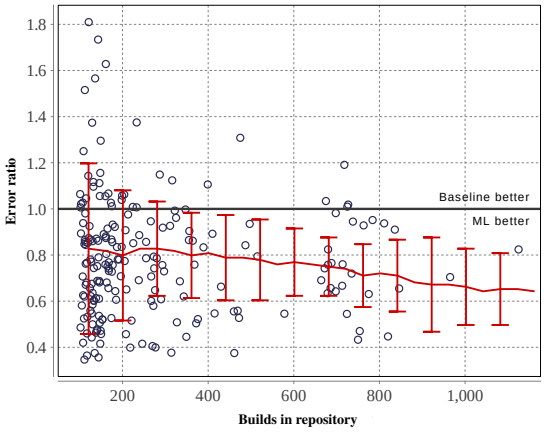

Figure 3.5: Error ratio over builds per repository, filtered ($T = 50$)

individual data points from these figures, such as outliers, cannot be correlated.

Figure 3.6 shows this comparison of performance for unfiltered and filtered datasets using various thresholds T. Here, we display several filtering thresholds and their performance outcomes. While the nominal performance increases slightly with additional filtering, the impact in the median case is not drastic (0.82 without filtering, compared to 0.78 for $T = 200$). The reduced number of outliers is accountable to the very fact that we are removing outliers in the domain range (below a certain file size), in turn removing those results as outliers from the value range.

Summarizing, despite seeming promising at first, filtering input data only increases prediction performance artificially by a small amount. We therefore did not perform such filtering.

3.4.2 Detailed Analysis of Resulting Models

In this section, we inspect the resulting (trained) ML models in detail. Each output variable (label) stems from two input variables, forming a three-dimensional system, and the resulting plot for the prediction function is a (non-flat) surface. In the following, we show two-dimensional graphs. For each graph, one of the input variables is shown on the horizontal axis, and the other one is reduced using projection, i.e., three dimensions are projected onto two dimensions.

We select four individual exemplary repositories (denoted as `A`, `B`, `C`, and `D`) to demonstrate the functionality of our ML prediction approach.

Examples of Good Fit

In Figures 3.7 and 3.8, `A` and `B`, respectively, are shown as examples of repositories with a good learning fit. Figure 3.7 shows the prediction for `A` regarding CPU utilization (as projected over file count) and Figure 3.8 shows the prediction for

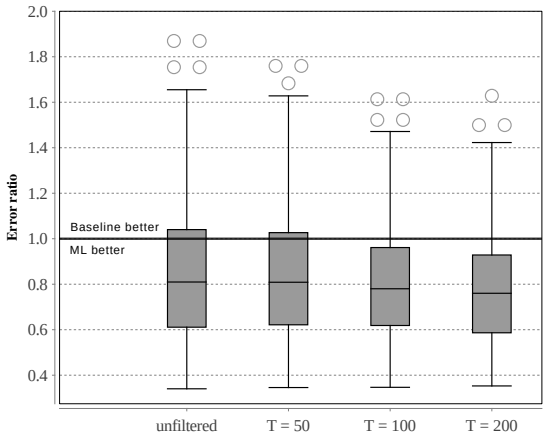

Figure 3.6: Error ratio of unfiltered and filtered data

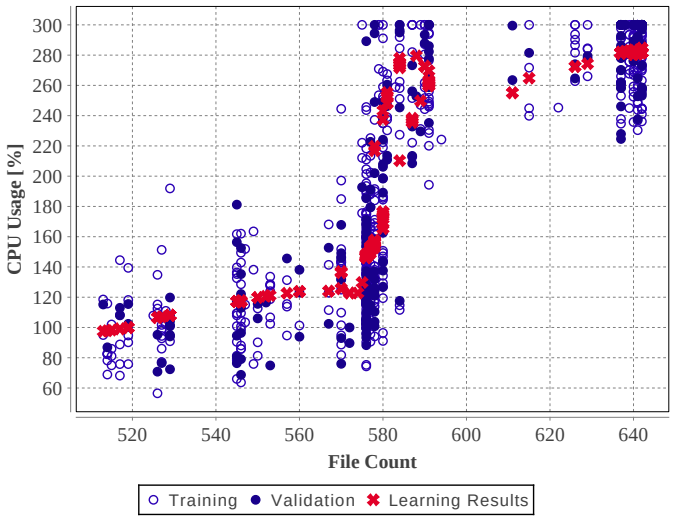

Figure 3.7: CPU utilization over file count for repository `A` (error ratio = 0.161)

`B` regarding duration (as projected over repository size). The error ratios are 0.161 and 0.171, respectively, which means that our approach yields a 84% and 83% decrease in error, respectively, compared to the baseline.

Input variables are shown on the horizontal axes, while output variables are shown on the vertical axes. Blue marks denote the dataset (hollow circles for training, opaque circles for validation), while red cross marks indicate the ML model predictions. Observing the red trace of cross marks, we see that the ML model has indeed trained to follow a curve corresponding to the training data. While the training data variance is high, the trace of learning results is generally close to an average value throughout the entire domain.

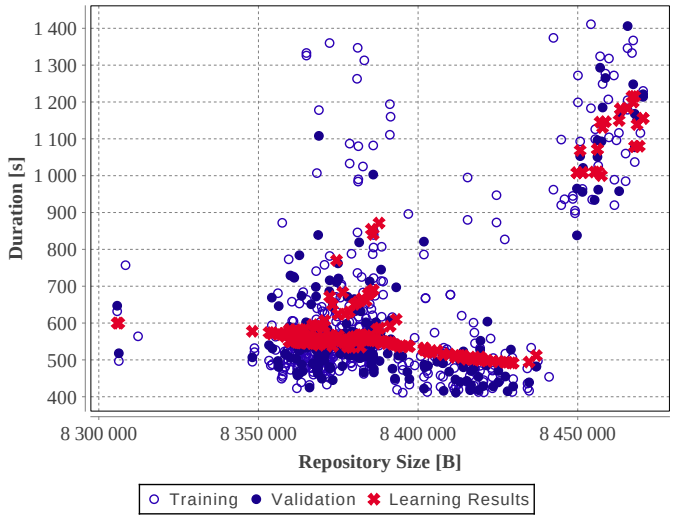

Figure 3.8: Duration over repository size for repository `B` (error ratio = 0.171)

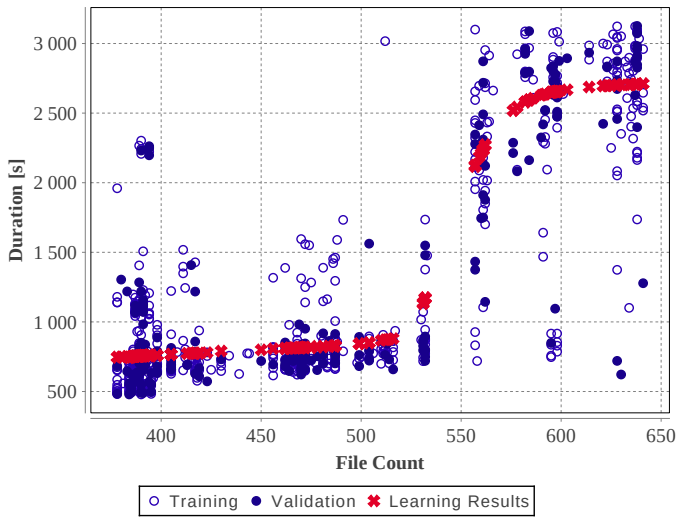

Figure 3.9: Duration over file count for repository `C` (error ratio = 0.267)

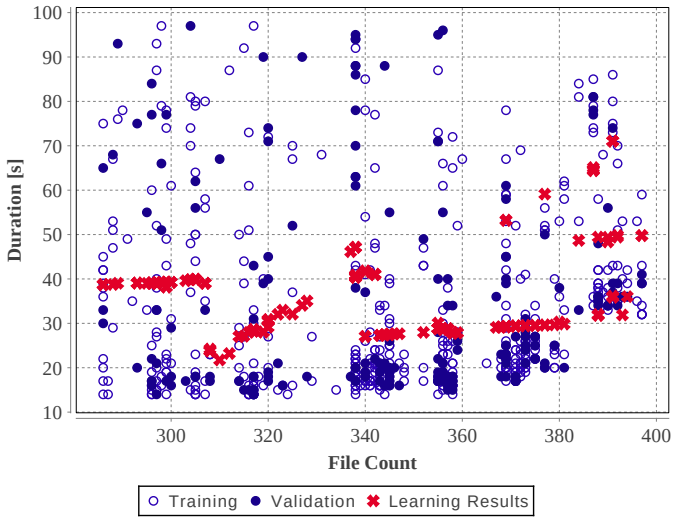

Figure 3.10: Duration over file count for repository D (error ratio = 1.682)

Another example of good fit is visible in Figure 3.9, where the predicted duration is shown over the file count for repository C. The ML model performs well compared to the baseline approach, and the error ratio for this repository is 0.267, indicating a 73% improvement over the baseline, despite the more scattered data for this repository.

Examples of Bad Fit

To also analyze examples of bad fit of our ML model, we observe the results of repository D in Figure 3.10. We see that due to the high variance of the input data, the ANN has not been able to find a fitting function. Even though the resulting estimation is not far off the training data, the error ratio is 1.682, i.e., the mean error is 68% higher than for the baseline approach, indicating that the baseline approach performs better than the ML approach.

3.4.3 Result Overview and Discussion

When analyzing in detail the graphs shown in Section 3.4.2, resulting from various runs of our approach on the dataset, we see examples of good training fit in Figures 3.7, 3.8, and 3.9, where the resulting reduction of prediction errors is 84%, 83%, and 73%, respectively. Figure 3.10 shows an example of bad training fit, where our approach yields a prediction error increased by 68%, compared to the baseline. We observe that high variance in the domain range severely reduces the ANN performance.

Our main concern is the distribution of instances with good fit and instances with bad fit in the results of our experiment. For this, we show the overall distribution of the performance of our approach in Figure 3.11. Here, the CDF of error ratios obtained with all repositories in the entire dataset is shown. We see

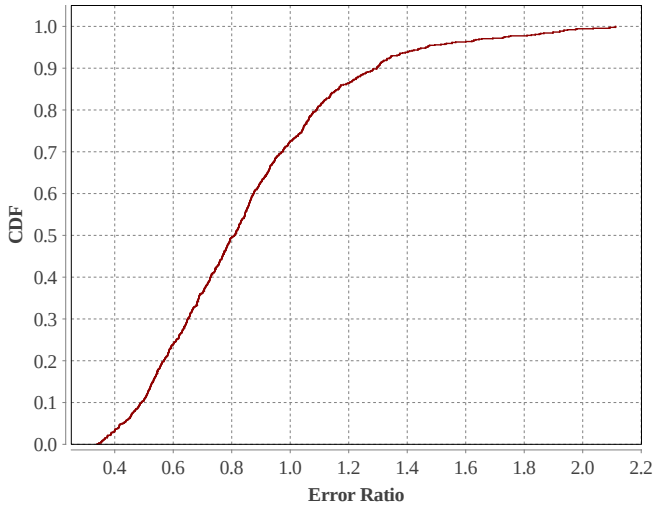

Figure 3.11: Performance of per-repository prediction

Table 3.4: Summary of aggregated results

	Approach	RMSD	Error ratio
Duration	Baseline	10.4	1.00
	ML (worst 5 %)	23.9	2.30
	ML (median)	8.3	0.80
	ML (best 5 %)	1.1	0.11
CPU	Baseline	12.7	1.00
	ML (worst 5 %)	31.8	2.50
	ML (median)	9.8	0.77
	ML (best 5 %)	2.3	0.18

that the median case (CDF = 0.5) exhibits an error ratio of 0.80. In other words, our approach yields a median 20% decrease in prediction error. Furthermore, the error ratio 1.0 (i.e., the point until which our approach performs better than the baseline approach) has a CDF of 0.72. This indicates that for 72% of all repositories, our proposed ML model outperforms the baseline approach (error ratio below 1.0).

Table 3.4 compares the performance of our ML predictor to the baseline predictor. In the table, we show the RMSD and the error ratio for both evaluated approaches. We show the 5% and 95% percentiles for the ML predictor approaches, as well as the median (50% percentile), to provide an overview of the result distribution. We see that in the median case, the duration (R_1) is predicted with a RMSD of 8.3 s, which, compared to the baseline RMSD of

10.4 s yields an error ratio of 0.80. The CPU utilization (R_2), in the median case, is predicted with a RMSD of 9.8%, yielding an error ratio of 0.77.

Limitations of our study can mostly be observed regarding the dataset used in the evaluation, and general applicability to other scenarios. The dataset stems from a public cloud-based CI service, and no information about the underlying hardware and software setting is known. Therefore, with this data, we cannot study the impact of different types of nodes on the resulting data, which would require a more extensive dataset with additional information about the used nodes. Nevertheless, we argue that the evaluation results show sufficient applicability of our approach in a typical scenario. The cloud-based nature of the dataset source together with the build processes used as example tasks in this chapter are typical for situations where cloud resources (in this case, CPU utilization and time) are served to clients in an *ad hoc* manner.

3.5 Related Work

Several approaches to predicting or estimating the utilization of resources for tasks to be provisioned using cloud-based computational resources exist, ranging from relatively simple mechanisms to complex ones. In this section, we present and briefly discuss these approaches.

Time Series Analysis

Sarkar et al. [227], similar to our approach, use analysis of historical time series, but employ a clone detection technique to determine whether resource access patterns have been encountered before. Gandhi et al. [101] use a hybrid approach of predictive and reactive provisioning, where the predictive approach works at coarse time scales (hours) and the reactive approach handles short-term peaks (seconds). Similarly, Caron et al. [58] use time series analysis and propose an auto-scaling algorithm. However, for [58], [101], and [227], the prediction capabilities are restricted, and no ML techniques are used, contrary to our approach, leading to an increase of required expert knowledge.

Scaling and Placement

Similarly to the provisioning or placement problem, the scaling decision problem is a main subject of numerous research efforts, especially in cloud computing. Jiang et al. [146] use a linear regression model to predict future values of resource utilization. The authors provide CPU, memory, I/O, and network bandwidth as examples for predictable metrics. In their evaluation, their approach is applied to predicting the number of requests. Based on this prediction, the system performs scaling operations. Caron et al. [58] also predict usage (request numbers), however, they use a pattern matching algorithm for this prediction. Dutreilh et al. [82] also seek to automate resource allocation (placement) by using reinforcement learning with Q-learning [261]. However, in contrast to this work, which only considers scaling, we provide an approach applying ML, based on which decisions including placement, scaling, and scheduling, can be reached.

However, as argued in Chapter 2, the scaling decision problem (leasing and releasing) alone is often not sufficient for efficient operation, as provisioning and placement of resources must also be considered using ML.

One group of such approaches is found in the field of BPMS, where recent work includes elastic BPMS [230]. This work uses predefined values for each task type, e.g., defining the amount of CPU a task is expected to utilize [231]. More recent work uses simple (non-trained) linear regression to predict resource utilization [135] or Mixed Integer Linear Programming (MILP) techniques to find an optimal provisioning (placement) of service instances [133]. Using our ML prediction, these approaches and scenarios can be refined and enhanced by more sophisticated resource utilization prediction.

Viswanath et al. [257] propose an approach similar to our work in that it employs neuron-based systems to predict resource utilization behavior, but focuses on the clustering and load distribution of this prediction, instead of the prediction of resource utilization of tasks themselves.

ML Prediction

To the best of our knowledge, two works come closest to the work presented in this chapter: Both the studies conducted by Mason et al. [186] and by Islam et al. [140] propose empirical prediction of resource utilization in the cloud. In addition, both works apply ANN models: Islam et al. use multi-layer ANNs and linear regression, while Mason et al. use a combination of ANNs and evolutionary algorithms called evolutionary ANNs. Islam et al. analyze TPC-W, a specification for benchmarking e-commerce scalability and capacity planning for non-cloud e-commerce websites [192], while both Mason et al. and the approach presented in this chapter use real-world data records for evaluation. However, both related approaches restrict the prediction to the overall CPU utilization of a host, while we predict the CPU utilization and duration of individual tasks. Thus, the results of this approach are more coarse-grained than the predictions yielded by the approach presented in this chapter.

3.6 Summary

In this chapter, we have shown how to employ methods from the field of ML to build prediction models for resource utilization from historical data. Using a real-world dataset collected from a cloud-based CI platform, we have evaluated our approach and shown that it indeed increases accuracy, as less prediction error is experienced, compared to the baseline approach. In the median case, our model predicts the task duration with an error rate of 0.80 (i.e., 20% less prediction error) and the CPU utilization of cloud tasks with an error ratio of 0.77 (i.e., 23% less prediction error). In the best 5% of cases, for the task duration, an error ratio of 0.11 is achieved (i.e., 89% less prediction error), and for the CPU utilization, the error ratio is 0.18 (i.e., 82% less prediction error).

CHAPTER 4

Failure Prediction in Business Processes

Business Process Management

BPM addresses the problem of how to design, analyze, configure, enact, and evaluate business processes [263]. A business process is defined by van der Aalst et al. as "*a generic software system that is driven by explicit process designs to enact and manage operational business processes*" [1]. In the last two decades, research efforts in the BPM field have resulted in a rich toolset covering all phases of the BPM lifecycle [1], however, with a strong focus on centralized and intra-organizational processes. In contrast, distributed and decentralized business processes have received comparably little attention [43].

Nevertheless, today's business processes are inter-organizational and distributed to a large degree, since companies need to collaborate in order to generate a desired output. Examples for distributed processes can be found in health care [184], manufacturing [229], or smart grids [219].

Event-Based Systems

One way to include a notion of distribution into business processes is by adopting basic concepts from the field of EBS [202], which defines a software architecture pattern based on events, i.e., state changes of process-related objects [201]. Instead of applying a request/response, *pull*-based messaging pattern, EBS decouple message producers and consumers by *pushing* events to receivers. As one prominent example, the publish/subscribe pattern is based on events which are sent from a publisher to subscribers [88]. Importantly, event messages are not aimed at a particular receiver. Instead, a notification service decouples producers from consumers, and delivers events whenever necessary. This allows separation of event-based communication from computation [202]. While EBS can also be centralized, distribution is usually seen as an inherent feature of modern EBS [162]. This applies both to the potential distribution of data to

be processed as well as the EBS itself, i.e., such a system can be distributed in order to allow horizontal or vertical scalability. With the advent of the IoT [12], a virtually unlimited number of potential event sources exist.

Events can be used to control and adapt distributed business processes during design time, change time, and runtime, or to simply exchange data between different process stakeholders. This includes events coming from IoT devices, but also data from business intelligence or any other event-producing system.

Context Events

One particular application area of events in business processes is their usage in order to predict potential failures during process execution. To the best of our knowledge, research on fault tolerance in business processes has so far focused on the exploitation of process-inherent knowledge, e.g., from process logs [23, 152], while only a limited amount of approaches consider data stemming from the context of a process, such as context events. Such events—sourced from IoT technologies or other data sources—can influence the outcome of a process, and should therefore be taken into account. Thus, we seek to examine the exploitation of events in order to find errors and predict potential failures during process execution. Based on such a prediction, it is possible to initiate according countermeasures in order to prevent a fault from causing an actual failure.

We investigate the exploitation of events in business processes by combining BPMS with EBS. Similar to the scenario considered in Chapter 3, we process labeled data. However, in this chapter, this data is represented as an event stream, which means that each data item is not independent of its predecessors. The labeled nature of the data again indicates the usage of ANNs, but the dependency of data items (events) on each other makes recurrent layers using Long Short-Term Memory (LSTM) an especially suited sub-type of ANNs [77]. We employ these ANNs to predict whether a running business process is likely to fail, and at which step. Since business processes often include interactions between various partners, which can significantly influence the amount and type of available context events, we additionally analyze the impact of inter-organizational processes on the prediction performance.

We provide an example scenario for a business process involving multiple partners. This scenario demonstrates the motivation for predicting a failure impacting the final process outcome before the failure's occurrence, in order to allow a timely reaction to such a failure.

> **Example Scenario.** A container with bananas is shipped from South America to Iceland. This shipment is part of a supply chain business process "Send goods from South American plantation to supermarket in Grundarfjörður". The container is equipped with sensors, which at some point of time identify a temperature exceeding limits, and therefore emit an according event. Thus, it is possible to derive that, with very high probability, the bananas are

somewhat rotten and therefore cannot be sold in the supermarket. Most importantly, this can be done before the container is actually opened at its destination in Iceland, and it is possible to order another shipment as a countermeasure to compensate for the likely faulty shipment process.

4.1 Fundamentals

In the following sections, we discuss fundamental topics for the content presented in this chapter, such as process orchestrations and collaborations, event streams, and the definition of failures in business processes.

4.1.1 Process Orchestrations and Collaborations

In our work, we consider both intra- and inter-organizational processes, and therefore investigate process orchestrations and collaborations. Both concepts entail the composition of services and other components necessary for a business process [94]. While a process orchestration represents the business logic of a single organization, a process collaboration involves multiple organizations collaborating to achieve a common goal [14]. In a collaboration, we mainly distinguish between three overlapping layers: (i) private processes, (ii) public processes, and (iii) a choreography model [43].

Private and Public Processes, Choreography Models [93, 159]

A private process represents the process orchestration—the business logic of one organization—and corresponds to its executable process [94]. In particular, it defines the relationship between tasks and characterizes both control and data flow. The internal logic and corresponding data is usually hidden from other organizations, e.g., due to confidentiality requirements [92]. In contrast, a public process represents the interface to other organizations and includes public tasks as well as the interaction activities from the perspective of one single organization. The public model logic and data are exposed to other involved organizations to enable the execution of inter-organizational business processes. Finally, a choreography model gives an overview of the collaboration between different organizations and defines the interaction flow between them. In particular, it describes the inter-organizational interactions, message formats, and content.

Relation to Example Scenario

In our example, we argue that a logistic business process includes private and public processes, as well as a choreography model: First, the initiator of the process—the owner of the supermarket—is interested in a timely and efficient shipment. Second, the shipping company has a business process of its own—its private process. Each organization's private process is hidden from the other organizations, and only the interface defined by the choreography model is visible, describing the total workflow involving all partners to achieve the common goal of shipping goods.

At runtime, each organization holds a set of process instances which run concurrently with the process instances of the collaborating organizations [96]. In order to correlate between them and to ensure proper interaction, we assume a global instance identifier. Without loss of generality, the latter is generated by one organization, i.e., the initiator, and transmitted to the other organizations for each new collaboration instance, allowing to correlate between exchanged messages and corresponding process instances.

4.1.2 Event Streams

Intrinsic and Context Events

A process task is a unit of work that is performed to complete a job, and involves a set of resources, i.e., humans or machines. When a resource performs a task, data is emitted in the form of events [160]. In this work, we distinguish between two main types of events. First, *intrinsic events* are emitted by process steps starting and finishing. They are intrinsic to the process model, consist of the process step and its attributes, or of a failure indication, if the given process step could not be finished successfully. Second, *context events* stem from external sources, including IoT devices, sensors, third-party business partners collaborating in the process, and other data sources. These events are not directly connected to the process model, but can be correlated to the process steps, e.g., using temporal or local proximity.

We define the entirety of all events as an *event stream*. A serialized form, either for transmission or for persistent storage, is an *execution log* or *event log* [211]. In some cases, the event log is part of the process log [211].

In our example, the event stream consists of both intrinsic events, i.e., the individual process steps like loading/unloading of containers, and context events, e.g., the readings from the sensors measuring the temperature of a banana container. Note that a context event does not need to be strictly and bijectively related to a certain process step, and may very well be used for failure prediction within multiple processes. For correlation, in our approach, it is assumed that the temporal co-presence of the process and the context event is sufficient. Nevertheless, an explicit correlation can be also expressed a priori to limit the event scope and increase system scalability (e.g., detecting high temperature is relevant only during the delivery process, not before or afterwards).

4.1.3 Faults, Errors and Failures

Differentiating the terms *fault*, *error*, and *failure* is significant in the context of failure prediction. In present literature, they have well-defined semantics [13]: A fault is the adjudged or hypothesized cause of an error. An error is the deviation of the system from its desired state. Finally, a failure occurs when the system is not able to deliver its output as it is supposed to, leading to an undesirable outcome. Therefore, a fault is the cause of the error, which may or may not

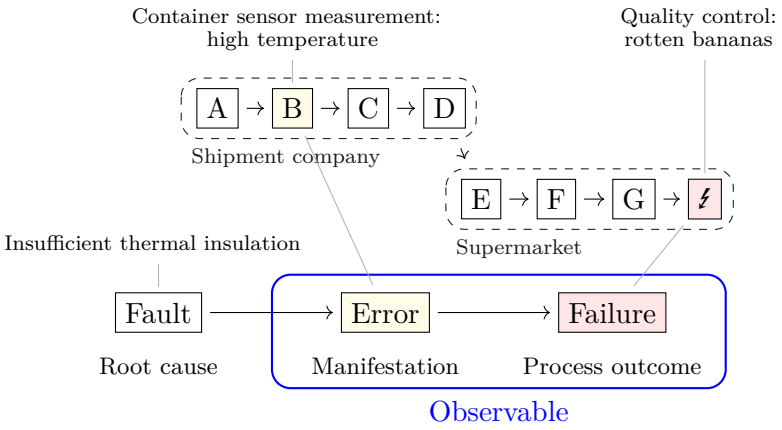

Figure 4.1: Example of faults, errors and failures within a process

manifest itself in a failure—it may remain unnoticed, without adversary effect, or it may be automatically corrected by the system. Conversely, the presence of a failure implies that during execution, at least one error (and therefore, one fault) has occurred [13].

Relation to Example Scenario

In order to associate these semantics with our scenario, we define their application to a BPMS. Figure 4.1 shows an example business process ending in a failure. In this example, the thermal insulation of a container represents the (potentially unobserved) fault, i.e., the root cause of the later failure. Subsequently, this leads to an error, manifesting itself during step B of the process (for instance, in a context event consisting of a sensor measuring high temperature), and can be detected by an observer. Finally, after steps C through G have passed, the rotten bananas arrive, and the overall process failure—the failure to deliver edible bananas to the supermarket—becomes evident.

Since according to its definition [13], the fault itself is generally not observable, we can only detect its earliest manifestation, the (first) error. It is noteworthy that time passes between the fault and the error, as well as between the error and the failure. In some processes, those delays may be long enough to initiate countermeasures if a failure is deemed likely already at the time the error occurs.

Without any prediction, an event constituting an error might not be noticed as such, and the failure only becomes evident upon its occurrence. However, adding a predictive element allows us to foresee a possible failure outcome at the first point in time it is detectable, i.e., when the first sign of an error occurs. In present literature, rule-based prediction is used to define the characteristics of errors [124, 276]. In our proposed architecture, these rules are substituted by a ML component.

4.2 Solution Overview

Integrating EBS and BPMS enables to control and adapt the execution of business processes at runtime by leveraging on both intrinsic and context events.

During the execution of a business process in a BPMS, the concrete services instantiated for the tasks contained in the business process are executed, and this execution generates intrinsic events. These events consist of task status changes, e.g., start and termination. With respect to a specific business process, other and more detailed intrinsic events can be defined (e.g., delivery suspended, machine restarted, network connectivity unavailable). Moreover, the service provider can enrich these events with non-functional and QoS information related to the service execution, such as the amount of resources or the monetary cost required to perform the task.

Furthermore, during its execution, a business process interacts with the environment (context) surrounding the invoked services. Therefore, we can identify external data sources, which, by generating context events, enrich the process execution with further information.

As mentioned in Section 4.1.2, context events must be correlated to the business process, either using temporal information, i.e., a sensor reading during the runtime of a process step, by local proximity, or using expert knowledge to define certain event sources as relevant for a given step. Note that we do not necessarily need to have information about causal relationships, i.e., it is not necessary to define that an excessive temperature reading indicates an upcoming process failure. Instead, we merely define that the temperature sensor measurement is happening during the shipment step. Although many data sources can be identified, the process of identifying the relevant ones, leading to the generation of valuable information, strictly depends on the specific business process and on the application that will exploit this data. For example, if we want to monitor the shipping process and predict the ability to deliver on time, relevant data might come from weather monitors (e.g., to detect the presence of snow or heavy rain), route monitors (e.g., to predict traffic jams), or the presence of human agents who can slow down the process.

Event-Based Failure Prediction

To show the benefits of integrating BPMS and EBS, our solution utilizes EBS data to perform Event-Based Failure Prediction (EFP) for the business process execution, and this prediction is performed by the *EFP component*. Specifically, the EBS hosts and executes the EFP component which predicts the probability of failures at runtime, i.e., during the business process execution. The EFP component automatically learns the failure model by leveraging on intrinsic and context events. Our solution is general enough to be able to predict failures related to functional and non-functional dimensions, i.e., it can identify an unsuccessful termination of the process, e.g., product not delivered (functional

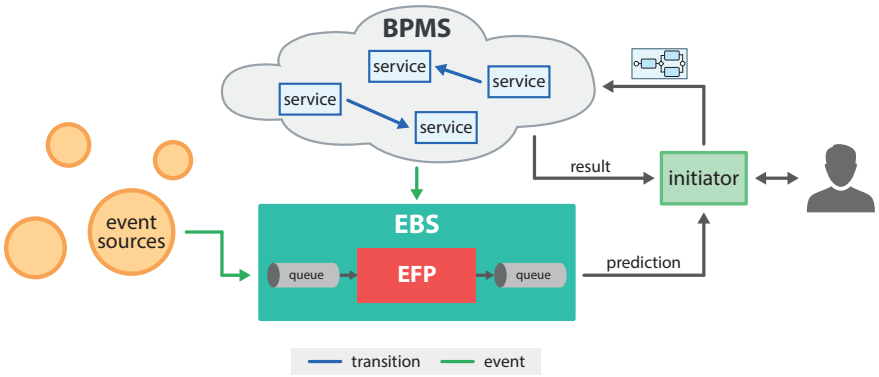

Figure 4.2: Proposed system architecture

failure) or a termination with unsatisfying quality requirements, e.g., the delivery of a damaged product (non-functional failure).

A conceptual representation of our solution is depicted in Figure 4.2. A user who wants to execute a business process interacts with an *initiator* component which is then responsible for launching the execution of the business process and the EFP component, and forwarding to the user any failure predictions emitted by the EFP component (during execution) as well as the business process execution result (upon finishing). When the user requests the execution of a business process, the initiator submits its description, expressed as a workflow, to the BPMS. At the same time, the initiator triggers the EBS, which, in turn, creates a new instance of the EFP component that will predict failures for the newly started business process. If necessary, the initiator informs external data sources of interest for the business process so that they can forward their context events to the EBS. Within the BPMS, each task of the business process is instantiated using a service that, aside performing its operations, generates intrinsic events. The amount and type of data transported by these events depend on both the self-monitoring capability of the service and other motivations (e.g., security, privacy, political). Within the EBS, all collected events are forwarded to the EFP component.

To reduce the coupling among the involved entities (i.e., EFP, services, event sources), the EBS uses a message queue system, where the distributed data sources publish intrinsic and context events. The EBS allocates a new queue for each business process execution. By subscribing to this queue, the EFP component receives all the events related to the business process as a continuous stream. Since the EFP component is event-driven, each new event can trigger a failure prediction. To learn and identify failures, the EFP component exploits an online learning approach based on ANNs, presented in detail in Section 4.3. As soon as a prediction is available, the EFP component publishes a new event on

an outgoing queue. Based on these events, the user can perform operations, e.g., adapt the business process instance, notify a client, or trigger the execution of a different business process. As proof of concept of our approach, we implement the EFP component capable of processing an event stream stemming from a running business process, and predicting likely failures at runtime.

So far, we have considered business processes in a generic manner, without accounting for the organizations involved in their execution. As stated in Section 4.1.1, apart from intra-organization business processes, we consider inter-organizational processes, the fulfillment of which involves the collaboration among multiple partners. The presence of multiple parties raises the issue of privacy. As discussed, some organizations might not share information about their private processes, thus limiting the visibility of their private events. In Section 4.4, we explicitly consider this critical point while evaluating the efficacy of the proposed approach.

4.3 Machine Learning Failure Prediction

The core idea of our approach is the assumption that process execution failures can be predicted based on context events, and that it is therefore possible to identify early deviations from the expected process behavior.

As outlined in Section 2.2, rule-based approaches have significant drawbacks, and we therefore use ML for the proposed failure prediction approach. Note that our solution can co-exist with expert-generated rules. For instance, present rules could support the initial training phase, during which the ML model might not produce meaningful output. After initial training, the ML model could be used to subsequently verify whether present rules are still valid, or have been made obsolete by concept drift.

4.3.1 Event-Based Failure Prediction Component

The EFP component consists of two interacting systems. The first system is responsible for the prediction itself: While a process is executed by the BPMS, and events are populated through the EBS, the EFP component is analyzing these events. Based on this analysis, the EFP component might indicate that with a certain probability, a failure is likely going to occur in a given future process step. Should such a likelihood arise, the EFP component fires an event to notify subscribers of the event queue. These subscribers can then react, e.g., by prompting the user and evaluating possible steps to counteract. The second subsystem is responsible for training. After each completed process execution, a trace of the recorded events (including all executed steps) is used to train the prediction model, increasing accuracy in subsequent runs. Since the model is

performing its predictions on an event stream, and does not need to store the entire dataset in memory, our solution constitutes an online learning approach.

Convolutional and Recurrent Layers

At the core of our solution, a specially designed ANN model is responsible for analyzing the stream of incoming data and performing failure prediction. In addition to the hidden layers discussed in Section 2.4 and used in Chapter 3, our network contains convolutional and recurrent layers [77]. Convolutional layers consist of neurons aggregating input from neurons which are semantically similar, and have proven useful, e.g., for facial recognition [187] or natural language processing [64]. In our case, this aids to reduce the network's sensibility to minor changes in temporal sequences of events. Recurrent layers introduce circles into otherwise acyclical graphs of neurons; in our case, we use LSTM layers [132]. This adds state information to the network, which is used to process temporal sequences of events. As stated in the introduction to Chapter 4, the combination of both convolutional and recurrent layers combines the power of both to support efficient training of temporal data by the ANN [77], which is crucial for processing event streams where each data item is not independent of its predecessors, as is the case in business process event logs.

Table 4.1 summarizes the parameters of the ANN model used in this chapter. For our ANN, we use parameters based on our work shown in Chapter 3. While most of the parameters remain the same, we adapt some parameters for this application. The amount of input and output layer neurons is higher due to the higher cardinality of input and output data, respectively.

In contrast to Chapter 3, we use the rectifier activation function [107], instead of tanh. The rectifier function has the advantage of a lower bound (*cutoff*), below which no activation occurs. It has been presented by Hahnloser et al. [115] and is commonly used in recent years for classification problems [168]. In contrast, both the tanh function and the very similar sigmoid (logistic) curve approach their lower bound but only reach it in infinity, i.e., some residual activation occurs, no matter how low the sum of weighted inputs is. Due to the fact that the problem approached in this chapter is a classification problem—whereas in Chapter 3, we considered a regression problem—the ANN used here profits from the cutoff of the rectifier function. This allows us to reduce the number of training epochs and the weight update momentum necessary for sufficient performance to 100 and 0.85, respectively.

4.3.2 Input and Output Structure

As described before, the ANN model is presented with input data both during training as well as during the actual prediction phase. In the case of training, we also provide output data (labels) to the network for supervised learning. The input data consists of the type of event as well as the data associated with the event, if any. As described in Section 4.1.2, we distinguish between intrinsic

Table 4.1: ML model parameters used in this chapter

Parameter	Value
Input layer neurons	40
Hidden layers	1
Convolutional layers	1
Recurrent layers	2
Hidden layer neurons	10 per output
Output layer neurons	20
Activation function	rectifier [107]
Learning rate	0.01
Minimization algorithm	SGD [40]
Weight initialization	Xavier [106]
Weight update	NAG [205, 206]
Weight update momentum	0.85
Training epochs	100

events, stemming from the process itself, and context events, stemming from the external context of the process.

In our notation, we denote the types of process-intrinsic events (i.e., the possible process steps) as $I_{\text{fail}}, I_0, I_1, \ldots, I_n$, where I_{fail} represents a failure in the current step, and the remaining symbols $I_0, \ldots, I_n$ represent all possible process steps. Context events, e.g., generated by sensors, are denoted as $C_0, C_1, \ldots, C_m$.

We therefore define the input vector for the ANN as shown in (4.1)–(4.3), where n and m are the number of intrinsic and context events, respectively, known to the system. $\text{Input}_{\text{Event}}$ is a binary vector consisting of one variable I_i for each intrinsic event type in the process and one variable C_i for each context event type. Depending on the type of the incoming event, either exactly one variable I_i is 1 for the intrinsic event I_i, or exactly one variable C_i is 1 for the context event type C_i in a given input row; all other variables are 0. Furthermore, $\text{Input}_{\text{Data}}$ is a vector containing the data associated with the event, if any. This data might include, for instance, the sensor reading from a temperature sensor. The cardinality k of $\text{Input}_{\text{Data}}$ depends on the type of event, i.e., which one of the variables $I_{\text{fail}}, I_0, I_1, \ldots, I_n, C_0, C_1, \ldots, C_m$ is 1.

$$\text{Input} = [\text{Input}_{\text{Event}}, \text{Input}_{\text{Data}}] \tag{4.1}$$

$$\text{Input}_{\text{Event}} = [I_{\text{fail}}, I_0, I_1, \ldots, I_n, C_0, C_1, \ldots, C_m] \tag{4.2}$$

$$\text{Input}_{\text{Data}} = [D_0, D_1, \ldots, D_k] \tag{4.3}$$

The output of the ML model is a classification of what the next step of the model will be (or whether it will be a failure). The process steps contained in the process model correspond to $I_{\text{fail}}, I_0, I_1, \ldots, I_n$. Therefore, the output is a vector giving, for each process step, the probability that this process step will be the next one. Note that for the sake of simplicity, we only regard one execution branch of a process. However, this does not limit the applicability of the proposed model to multiple processes running in parallel. Following this, we formulate the output vector structure as shown in (4.4).

$$\text{Output} = [\widehat{I_{\text{fail}}}, \widehat{I_0}, \widehat{I_1}, \ldots, \widehat{I_n}] \tag{4.4}$$

4.3.3 Formalization Model

In order to formally describe the underlying problem and our approach, we introduce a model for reasoning about the predictions of process outcomes. To this end, we build upon the model of Probabilistic Automata (PA) [215, 225]. PA are a generalization of a Non-Deterministic Finite Automaton (NFA), where instead of a membership function determining which states can be reached with which input, these binary values are substituted by probabilities. Since we deal with likelihoods, we use the generalization provided by PA instead of NFA. Formally, a PA consists of the following attributes [215]:

- A finite set of states Q.
- A finite set of input symbols Σ.
- A transition matrix P.
- An initial state[1] $q_0 \in Q$.
- A set of states $F \subset Q$ which are defined as final states.

In our model, the set of states Q is the set of process steps, including the *failure* state q_{fail}, representing a failure in a business process. For the set of input symbols Σ, we use the set of input and context events read by the predictor. Our initial state q_0 is the start state of the business process. The set of final states F is equal to the set of end states of the business process at hand. By definition, the failure state is also a final state, i.e., $q_{\text{fail}} \in F$.

The transition (stochastic or probability) matrix P determines the probability for the automaton to enter another state, given a current state and an input.

[1] Note that some literature uses a *distribution vector* for determining the initial state [225] instead of a fixed single state q_0. We use a single initial state, since the process model can be assumed to have a fixed initial state, and this simplifies the definition without reducing expressive power.

A common notation in the definition of PA is $p_j(q_i, x) \in P$, denoting the probability for the PA, with the current state q_i and given the input x to enter state q_j [225]. Naturally, the sum of the probabilities for all subsequent states of a state q_i and an event s is 1, since it is certain that *some* state must be reached, as shown in (4.5).

$$\forall q_i \in Q : \sum_{q_j \in Q} p_j(q_i, x) = 1 \tag{4.5}$$

In our model, the probability matrix P is not a fixed matrix. Instead, whenever a prediction of the following step is required, the previously described ANN is invoked, yielding probability values for all possible next steps. In other words, a row of the transition matrix is returned, as shown in (4.4).

We now define that, at any point in time during the execution of a process, there is an event trace T, which consists of all so-far recorded events (including intrinsic events, i.e., state changes, and context events). We argue that the current process step is deducible from this event trace by merely searching for the last recorded intrinsic event indicating a process step, and define that step as q_i. In order to predict the subsequent process steps to deduce whether a failure might occur, we are interested in the probabilities for the process to continue with a certain step q_{i+1}. As discussed, instead of a fixed transition matrix to determine the probable next step q_{i+1}, we use the ANN which returns, for each possible step $q \in Q$, the probability for this step. We denote the application of the ANN model onto a given step trace T (the invocation of the model with T as its input) as $\widehat{X}(T)$.

4.3.4 Probability Traversal

Following the previous definitions, we describe the further processing of predictions by the predictor component. We have already defined an event trace T, which denotes the already-recorded (historical) events and stems from a running process instance. We invoke the previously discussed ANN onto T, yielding a row vector out of the transition matrix; we call this vector P_T, as shown in (4.6).

$$P_T = \widehat{X}(T) \tag{4.6}$$

For each step $q \in Q$ (corresponding to the states of the PA), $P_T(q)$ yields the probability of the process to continue with this step q. This probability for a single step is called *step* probability. The sum of all step probabilities for a given event trace T is 1, since *some* step, possibly q_{fail}, is certainly going to be the subsequent one.

For instance, Figure 4.3 shows a trace of events T, containing the events $A \to B \to C$. Possible subsequent events, including failures, are shown together

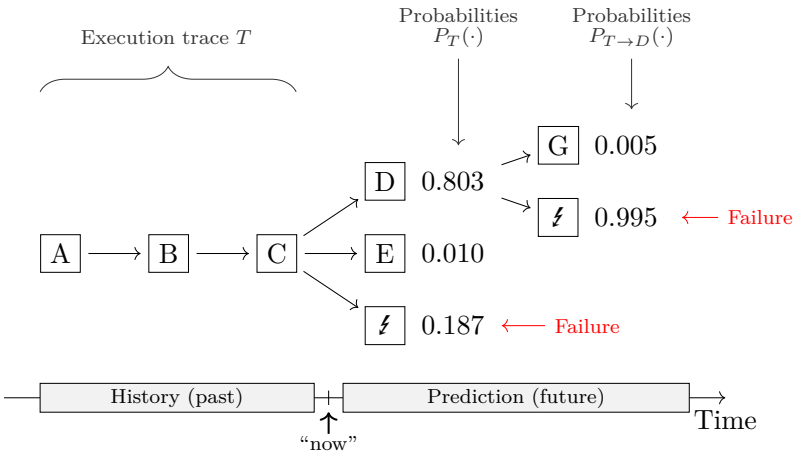

Figure 4.3: Example event tree for the trace $A \to B \to C$

Table 4.2: Non-zero probabilities for next event, corresponding to Figure 4.3

q	$P_T(q)$
D	0.803
E	0.010
q_{fail}	0.187
Σ	1.000

with their probabilities. Elements E and G are defined as final states. The probability vector P_T has been obtained by invoking the ANN model, yielding $P_T = \widehat{X}(T)$. The values for P_T are shown in Table 4.2 (events with a probability of zero are emitted for brevity).

Similarly, this traversal is also performed for subsequent steps. In this manner, we traverse the entire space of possible subsequent steps, and for each possible step q, re-evaluate all following possible steps. This traversal is done until an end step is reached, at which point the traversed trace is recorded, together with its total probability and its outcome.

The total probability is the conditional probability of a given trace T to be followed by the future step sequence T', and is denoted as $\mathbb{P}(T \to T')$. The outcome defined as $\Omega(T')$ denotes how the sequence T' ends, and is either *end*, for an orderly finished process, or *fail*, if a failure occurred, i.e., if q_{fail} was reached. The probability $\mathbb{P}$ is defined as shown in (4.7)–(4.10).

$$\mathbb{P}(T) = 1 \tag{4.7}$$
$$\mathbb{P}(A \to B) = \mathbb{P}(A) \cdot \mathbb{P}(B) \tag{4.8}$$
$$\mathbb{P}(A \to q) = \mathbb{P}(A) \cdot \mathbb{P}(q) \tag{4.9}$$
$$\mathbb{P}(q) = P_T(q) \tag{4.10}$$

In (4.7), we define the total probability of the original event trace T as 1, since the event trace has already been recorded, and thus its occurrence is certain. In (4.8), the total probability of an event sequence A followed by another event sequence B is defined as the product of the total probability of A and the total probability of B. Similarly, in (4.9), we define that the total probability of an event sequence followed by an event is, again, the product of both probabilities. Finally, (4.10) defines that the probability of a single element is equal to its step probability (i.e., the invocation of the ANN is formalized).

Naturally, the sum of the total probabilities of all possible event sequences following a given trace T is 1, since *some* event sequence, possibly one where the outcome is a failure, will eventually be the resulting sequence of the process.

To formalize our traversal, we define the TRAVERSE function of an event sequence T, which aggregates the results yielded by $\widehat{X}(\cdot)$, as shown in (4.11), where VISIT$(\cdot)$ is the function generating a set of resulting sequences, based on this event sequence $T \to q$.

$$\text{TRAVERSE}(T) = \bigcup_{q \in Q} \text{VISIT}(T \to q) \tag{4.11}$$

The function VISIT$(\cdot)$ is defined as shown in (4.12). As we can see, the function, upon encountering a non-final element q, invokes the TRAVERSE function again using the concatenation of T and q, i.e., $T \to q$.

$$\text{VISIT}(T \to q) = \begin{cases} T \to q, & \text{if } q \in F \\ \text{TRAVERSE}(T \to q) & \text{otherwise} \end{cases} \tag{4.12}$$

Table 4.3 shows the resulting sequences of the example process, together with their probabilities $\mathbb{P}$ and outcomes Ω.

Putting the definitions together, the predictor component can, at any given point in time during the execution of a process, use the trace of already-recorded events T, and by invoking TRAVERSE(T), build a list of possible future event sequences, along with their probabilities $\mathbb{P}$ and outcomes Ω. In the example at hand, the most likely scenario is the continuation of the process to D, where a failure will occur, and the likelihood for this scenario, compared to all possible scenarios, is 0.799.

Table 4.3: Probabilities and outcomes of Figure 4.3, with $T = A \to B \to C$.

T'	$\mathbb{P}(T \to T')$	Ω(T')
$D \to q_{\text{fail}}$	0.799	*fail*
q_{fail}	0.187	*fail*
E	0.010	*end*
$D \to G$	0.004	*end*
Σ	1.000	

4.3.5 Search Space Optimization

For large process models, the search space defined by the recursive function TRAVERSE can become too large to be processed in an online matter, i.e., during the process execution. If the processing time increases, the outcome might not be predicted in time. To mitigate this, we propose several ways of limiting the search space of the recursive algorithm.

Process Model Correlation Since the underlying predictor uses an ANN model to predict subsequent events (including steps) in a process, the results of this prediction might include events which are not possible in the current state. For instance, if there is no control flow relation between steps C and G in the example shown in Figure 4.3, and the last event in the recorded event sequence is C, G is not a possible subsequent event, and this part of the tree does not need to be explored. With a naïve search, however, the ML model might still consider this impossible sequence, and it could even be predicted as the *most likely* sequence. This is especially the case during the phase of initial training, or in unusual or novel event sequences. The reason for this behavior is the fact that the ML model itself does not take into account the process model being executed. Therefore, we introduce a stopping condition for the traversal. This condition filters out event combinations that are not possible according to the underlying process model.

Probability Limit In a similar manner as with the previously discussed event sequences impossible according to the process model, we also disregard highly unlikely events. Our solution introduces a probability parameter `MIN_PROBABILITY` which is applied to the total probability of the entire path. In other words, a possible event branch is not traversed further if its current total probability falls below a threshold.

Depth and Breadth Limit In addition to a minimum probability, we introduce the parameters `MAX_DEPTH` and `MAX_BREADTH` defining the upper bound for depth and breadth within our search. Limiting the search depth

is based on the fact that predicting events which are too distant in the future may become decreasingly meaningful. Limiting the search depth numerically is working together with limiting the probability to reduce the search space.

The limits represent hyper-parameters of our solution and are used to maintain an upper bound on the traversal runtime. The primary goal of this bound is to avoid infinite traversal in processes with cycles (e.g., loops).

Listing 1 shows a consolidated, algorithmic form of all the calculations described above. While the function RECURSE contains the main (recursive) logic, TRAVERSE executes the initial call for RECURSE. The RECURSE function takes two parameters: T, which is the process trace which is assumed to be already known, and P_{curr}, which determines the total probability until the current step, as defined in (4.7)–(4.10). Naturally, TRAVERSE initializes this total probability with 1, because the recorded history of steps has already happened, and therefore has a probability of 1, as defined in (4.7). Lines 10 and 11 represent the depth and probability limits, respectively. Line 12 calls the ANN in order to obtain a vector of probability values for each possible subsequent step. This vector is sorted in line 13 by probability.

The loop in lines 15–33 iterates over the n most likely subsequent steps, where $n =$ `MAX_BREADTH` (ensured by line 18, the breadth limit), and checks whether they are an end state, a failed state, or a normal process step. For end states, lines 20–23 add an element to the result vector. Similarly, lines 25–28 handle failures, which are treated similarly to end states, since they also indicate a potential outcome of the business process.

For all non-failure and non-end states, lines 30–31 represent the recursive call to RECURSE, which concatenates T and the current event e, creates the resulting total probability of this partial (unfinished) trace, and performs another calculation as described above.

4.4 Evaluation

This section presents an empirical assessment of the proposed approach following the *single-case mechanism experiments* validation method [265]. The evaluation was conducted on both a real-world dataset from the finance domain, and a synthetic dataset created from a realistic distributed manufacturing process. The datasets were preprocessed and then used for training ML models. This allows us to assess the performance of the ML models in detecting failures. The conducted experiments prove the applicability and feasibility of combining EBS and distributed business processes in a real-world scenario.

Listing 1 Bounded process tree traversal

```
1: procedure TRAVERSE(T)
2:     Input: T: The event trace from the currently running process
3:     Output: Possible process outcomes, with probabilities
4:     return RECURSE(T, 1.0)
5: end procedure

6: procedure RECURSE(T, P_curr)
7:     Input: T: An already-fixed event trace T = [T_0, ..., T_i]
8:     Input: P_curr: Total probability of the current event trace
9:     Output: Possible further process traces from T_i, with probabilities
10:    if |T| > MAX_DEPTH then return []                          ▷ Depth Limit
11:    if P_curr < MIN_PROBABILITY then return []                 ▷ Probability Limit
12:    Pred ← GETANNPREDICTIONS(T)                                ▷ Fetch predictions from ANN
13:    sort Pred by Pred.probability descending
14:    Breadth ← 0
15:    for all e ∈ Pred do
16:        NextStep ← e
17:        NextStepProbability ← Pred[e]
18:        if ++Breadth > MAX_BREADTH then continue               ▷ Breadth Limit
19:        if e is end state then
20:            Trace ← T ⊕ e                                      ▷ ⊕ is the concatenation operator
21:            Probability ← P_curr · NextStepProbability
22:            Outcome ← "End state e reached"
23:            add < Trace, Probability, Outcome > to Result
24:        else if e is failure then
25:            Trace ← T ⊕ failure
26:            Probability ← P_curr · NextStepProbability
27:            Outcome ← "Failure in e"
28:            add < Trace, Probability, Outcome > to Result
29:        else
30:            NextPossibilities ← RECURSE(T ⊕ e, P_curr · NextStepProbability)
31:            add NextPossibilities to Result
32:        end if
33:    end for
34: end procedure
```

4.4.1 Datasets

In order to conduct experiments to evaluate the feasibility of our approach, we explored various datasets of different domains, and studied their usability and suitability for the evaluation of our approach.

Dataset Criteria

The search for an appropriate dataset for the experiments was conducted with respect to the following criteria: Whether the dataset is publicly available to the research community (PU), event-based (EB), correlated with business process models (BP), well-documented (DO), whether it contains context events (CE) and failures (FA), and whether it stems from an inter-organizational process (IO). The two-letter abbreviations are later used in Table 4.4.

A multitude of data sources have been examined from projects, research challenges, and other public data sources. In particular, datasets from the Business Process Intelligence Challenge (BPIC)[2] and the Kaggle Competition[3] were taken into account. Platforms for competitions in the domain of data science and ML, such as Kaggle, are naturally a valuable source for test data. Public directories for datasets are also available[4]. Often, these datasets are community-created[5]. However, due to the fact that we aim at considering context events, i.e., events not directly related to the core business process, the spectrum of usable datasets is significantly narrowed.

In the following, we briefly show the most promising dataset candidates and discuss our selection. We have identified a number of possible datasets such as the data from Bosch Production Line Performance[6]. In the underlying scenario of this dataset, parts move through the production lines of a manufacturer. While this happens, features are measured and recorded. The dataset has been anonymized with respect to the names of the features. For each measurement, the part is evaluated, and if it fails quality control, this is recorded and the part is dismissed. The dataset is highly imbalanced with regard to the actual class (i.e., failures or passes), and contains a very high number of features (over 12,000). While this dataset has the advantage of being rather large, the drawback is that no related business processes are defined. Furthermore, due to the per-part nature of the data, no business process can be mined as neither a stream of events, nor temporal correlation between measurements are available.

Another prediction-centered dataset stems from the Transport and Logistics Case Study (Cargo 2000)[7]. The latter includes events related to messages sent

[2] https://www.win.tue.nl/bpi/doku.php?id=2017:challenge
[3] https://www.kaggle.com/
[4] https://www.springboard.com/blog/free-public-data-sets-data-science-project/
[5] https://github.com/caesar0301/awesome-public-datasets
[6] https://www.kaggle.com/c/bosch-production-line-performance
[7] http://s-cube-network.eu/c2k/

within Cargo 2000, now known as Cargo iQ[8]. As shipments travel through segments of their transport (e.g., transfers between flights, airlines), their routes are recorded. This dataset has been sanitized with respect to erroneous or incomplete messages. The business process related to this dataset has been already used in research [97] and might have been relevant to our approach. However, despite the inter-organizational context, the provided dataset as well as the corresponding process are solely related to the freight-forwarding company and not to the entire process collaboration. Neither the private processes of the other involved organizations nor their respective data are available. Furthermore, as the dataset has been sanitized, it is difficult to identify failures to predict.

A third candidate for our evaluation is the Commodity Flow Survey (CFS) dataset[9]. It contains data about 4.5 million shipments. The data describes various attributes of the shipments, e.g., the source and destination of the shipment, type of commodity, whether or not the shipment requires temperature control during transportation, value, weight, modes of transportation, or hazardous materials. However, similar to the Bosch dataset, it consists of a series of single, independent entities. From this, it is difficult to create an event stream, since no temporal relation is given between the data entries.

Finally, the fourth candidate is taken from the BPIC datasets, various of which have already been used in research, e.g., [65, 90]. We regard the BPIC 2017 dataset[10]. It consists of an event log stemming from a Netherlands-based financial institute issuing personal loans to applicants. The process from which the data stems is not explicitly defined in the dataset, but can be mined using a process miner; e.g., ProM [221]. It contains enough variety in its events to be separated into intrinsic and context events, as described later in Section 4.4.2.

Dataset Selection

Table 4.4 summarizes the datasets with respect to the specified criteria, where check marks indicate fulfillment, and check marks in parentheses indicate partial fulfillment. Overall, no dataset satisfies all criteria, but the Cargo 2000 and the BPIC 2017 represent good candidates. Overall, we have selected the BPIC 2017 dataset, as it satisfies most of the aforementioned requirements, with the exception that no explicit context events (CE) and failures (FA) are provided in the dataset, and it is not based on a distributed business process (IO). We tackle the two first limitations by enriching the dataset as described in Section 4.4.2. For the third limitation, we decided to address the issue by using an additional, synthetic dataset, and use it in separate experiments to evaluate the inter-organizational aspect of our solution. The synthetic dataset is generated from an artificial, but realistic process collaboration. It was inspired by the CFS and the Cargo 2000 datasets.

[8] http://www.iata.org/whatwedo/workgroups/Pages/cargo2000.aspx

[9] https://www.census.gov/econ/cfs/pums.html

[10] https://data.4tu.nl/repository/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b

Table 4.4: Results of dataset evaluation

	PU	EB	BP	DO	CE	FA	IO	Comment
Bosch	✓			✓		✓		No event stream
Cargo 2000	✓	✓	✓	✓				No collaboration
CFS	✓			✓				No event stream
BPIC 2017	✓	✓	✓	✓	(✓)			Selected dataset

For our evaluation, we therefore opt for two datasets: (i) one from a real-world data source (BPIC 2017) to prove the applicability of the approach, and (ii) a synthetic dataset to show its feasibility in a distributed setting.

4.4.2 Dataset Pre-Processing

Due to the limitations of the real-world BPIC 2017 dataset with regard to criteria (CE), (FA), and (IO), data pre-processing is required.

Real-World Pre-Processing

First, we aim to achieve an event stream consisting of intrinsic and context events. As described before, the event log contained in the real-world dataset is taken from the application process for personal loans from a Netherlands-based financial institution. The log consists of three different types of events: *application* state changes, which have event names starting with `A_`, *offer* state changes, starting with `O_`, and *workflow* events, starting with `W_`. A loan application within the process may produce one or more offers. One of the offers occurring within an application may be accepted. In this case, the entire application process is finished. However, if no offer is accepted, the application process is canceled. The application and offer events represent this process. The process events represent the necessary actions to be taken within the financial institution. Since the core business process is the application for loans, we select all application events (`A_`) as process events. All remaining events (`O_` and `W_`) are used as context events.

Second, due to the lack of a process model provided along with the dataset, we apply data mining techniques to mine such a model. From the application events, we create a business process by using the Inductive Miner technique [169]. Using this technique is common in the field of process mining [125], and guarantees a certain level of *rediscoverability* [169]. We use the ProM tool[11] for process mining. The resulting process model is shown in Figure 4.4.

Finally, since the BPIC 2017 dataset does not contain any failures, we decide to select events as failures. In the dataset, around 35% of all loan application processes end with the process step `A_Cancelled` (as mined from the event log). This means that the application was canceled, e.g., because of missing

[11] `http://promtools.org/`

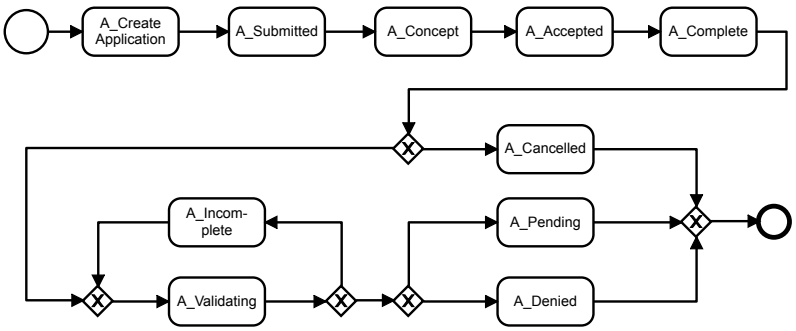

Figure 4.4: Process model mined from the real-world dataset

information, denial from the financial institution, or withdrawal by the applicant. Since this represents a failure to finalize the loan, we define this state as a failure, i.e., all loan processes ending in this state are treated as failed. No further injection was required for the real-world dataset.

Synthetic Pre-Processing

For the synthetic dataset, we opt for the generation of an artificial but realistic process collaboration. Data generation is inspired by the aforementioned CFS dataset. This dataset contains information on domestic freight shipments in different domains such as manufacturing and wholesale. Data include type, origin, destination, transport mode, and other shipment attributes. The dataset, however, includes one single event type rather than a stream of different event types, with neither time stamps nor correlation to process tasks or partners (no cases nor traces). Therefore, we define a collaborative process example of a supply chain scenario where goods are ordered, manufactured and shipped to the end client [95], similar to our example scenario. The scenario involves six process partners, i.e., a bulk buyer, a manufacturer, two suppliers, a special carrier and a middleman.

For each process partner, *private* and *public* tasks as well as interactions (through message exchanges) are defined. For each interaction, an Extensible Markup Language (XML) template specifying the data elements to be exchanged is created. The latter ensures *consistency* of data instances for the simulation. Indeed, within one execution of the entire process collaboration, the data required by one partner task might depend on the output or data of another partner task. Therefore, it is important that the generated data is consistent, even though it is produced randomly. For instance, the delivery date of an item by the *carrier* must not exceed the delivery deadline specified by the *bulk buyer* and transmitted to the *manufacturer*. In total, the collaboration contains 48 tasks distributed over the partners, of which 15 are interactions. Also, 20 data instances of message templates have been generated.

The process collaboration is simulated using the Cloud Process Execution Engine (CPEE) [243] in a distributed way, where each process partner is executed

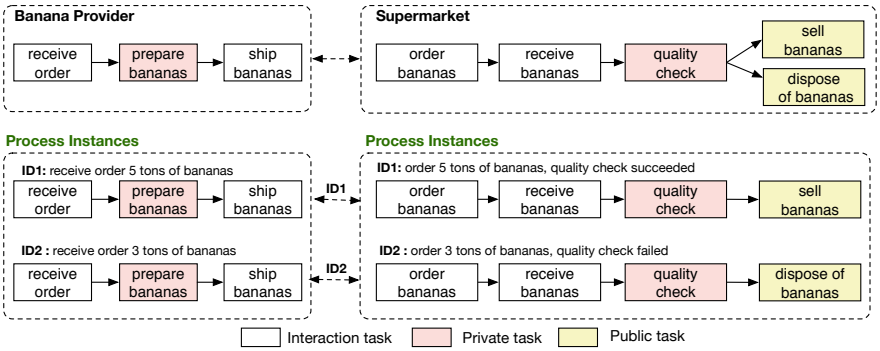

Figure 4.5: Collaborative process instances used in the synthetic dataset

separately on a different CPEE instance. The CPEE has been chosen because it provides an efficient, flexible and lightweight way of executing distributed workflows, while its modularity allows us to collect the events.

An asynchronous mechanism is implemented to correlate the messages of different partners. For this purpose, a global instance identifier is defined and exchanged through messages. The latter is primordial for process partners to correlate a received message with the correspondent process instance. We also distinguish between a process instance and a *collaboration instance*. While the former represents the instance of one single process, the latter refers to one execution of the entire collaborative process.

For example, Figure 4.5 describes two interacting processes, a banana provider and a supermarket. Each process is executed multiple times and each instance of the banana provider process must be correlated with the corresponding instance of the supermarket process. The instance identifiers ID1 and ID2 are specified in all message exchanges to ensure that data from one partner instance will be consumed by the right partner instance. A correlator (not shown in the figure) associates a message with the corresponding process and task instance (e.g., a message *5 tons banana order* with task *receive order* of instance id ID1). The latter can be centralized or distributed.

The CPEE enables easy collection of events including control and data flow as well as transactional information about the tasks. All events are stored in an Extensible Event Stream (XES) file [139], which is based on XML. The mechanism used to collect execution data from the CPEE also allows an easy collection and integration of events sent by a context event provider other than the CPEE [242]. This can be important for the prediction, as some process tasks might depend on a context event provider, e.g., sensor sources external to the CPEE. With respect to the previous example, Listing 2 shows an example of an event of type `order_banana` from the automatically generated XES file.

Listing 2 Example of an XES event log

```
<log xmlns="http://www.xes-standard.org/" xes.version="2.0"
  xes.features="nested-attributes">
<extension name="Time" prefix="time"
  uri="http://www.xes-standard.org/time.xesext"/>
<extension name="Concept" prefix="concept"
  uri="http://www.xes-standard.org/concept.xesext"/>
<extension name="Organizational" prefix="org"
  uri="http://www.xes-standard.org/org.xesext"/>
<extension name="Lifecycle" prefix="lifecycle"
  uri="http://www.xes-standard.org/lifecycle.xesext"/>

<trace xmlns="http://www.xes-standard.org/">
 <string key="concept:name" value="Instance 487"/>
 <event>
  <string key="concept:name" value="order banana"/>
  <string key="concept:instance" value="http://cpee.org/~demo/corr/corr.php"/>
  <string key="id:id" value="a4"/>
  <string key="lifecycle:transition" value="complete"/>
  <list key="data_received">
   <string key="result">
    <order_banana>
     <id>m11_1</id>
     <quantity>2000 tons</quantity>
     <delivery_deadline>2016-12-30<delivery_deadline>
     <date>2016-12-18<date>
     <address>California<address>
    </order_banana>
   </string>
  </list>
  <date key="time:timestamp" value="2016-12-15T15:58:37+01:00"/>
 </event>
</trace>
</log>
```

Failure Injection

We perform failure injection by artificially creating faults, which lead to errors, ultimately generating failures. We define and use the following three major fault types for injection into the synthetic dataset.

Step-indicated faults For this type of faults, the process shows a sequence of steps which is characteristic for the given fault. For instance, a fault may cause an Exclusive Or (XOR) gateway to proceed with a different process step than it would, had the fault not occurred.

Event-indicated faults Faults manifesting themselves only through certain events, but not through different process steps executed, are called event-indicated faults. For instance, a sensor measuring the temperature of a container of bananas may sense the violation of certain temperature limits and fire an event. In our model, this corresponds to a context event being fired during the process execution.

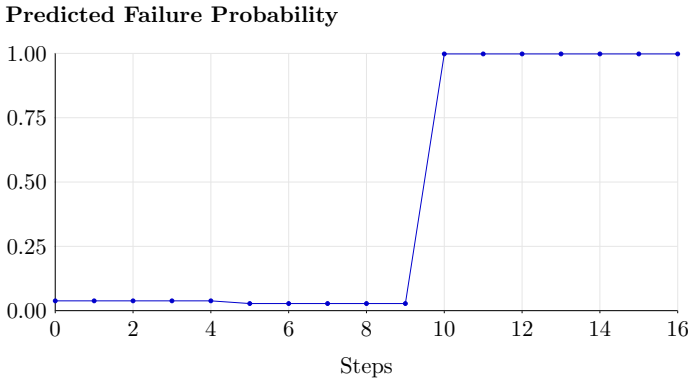

Figure 4.6: Example of an execution timeline

Data-indicated faults Certain faults are only indicated by the actual data associated with certain events. Considering the previous example, a temperature sensor may be recording temperature as events, regardless of whether a limit has been exceeded or not. In this case, the firing of the event itself does not correspond to a fault on its own. In fact, its associated data (the measured temperature) is the determining factor of whether a fault has occurred.

We inject faults randomly, using a given *fault injection rate*. This rate is varying, and part of our parameter sensitivity analysis in Section 4.4.3. We randomly select one of the three aforementioned fault types when injecting a fault. According to fixed fault-error-failure combinations, we add or change the corresponding events to reflect the errors, and measure the system's ability to properly predict failures stemming from these errors. We feed the event streams with injected faults through the ML predictor component, and, for each injected fault, record whether and at which point in time the failure is predicted. Figure 4.6 shows an example of such an evaluation run, showing that the injection of a fault which is exposed as an error at step 10 is detected by the predictor. The well-trained predictor immediately changes from indicating almost certain success (0.0276) to almost certain failure (0.9980) for step 16.

4.4.3 Experiments

The goal of our evaluation is to show that using our EFP component on the two selected datasets indeed yields useful prediction results. As described later in Section 4.5, our approach is novel in integrating external data sources and a common data format, while using an approach based on ML and taking into account the visibility of event data in an inter-organizational settings. Since due to this novelty, a reference baseline to compare our results to is not available,

Table 4.5: Confusion matrix for real-world dataset

	Classification		
Class	Positive	Negative	Σ
Positive	1051.14	28.04	1079.18
Negative	153.76	1917.95	2071.71
Σ	1204.90	1945.99	3150.89

Table 4.6: Performance metrics for real-world dataset

Metric	Mean	σ
Precision	0.873	0.014
Recall	0.971	0.008
MCC	0.879	0.012

we perform extensive experimentation to show the feasibility and applicability of our approach, and to provide a baseline for future research.

Methodology

Our experiments all involve running an instance of our EFP component, feeding the evaluation datasets into it, and recording its performance in predicting the probabilities of failure for each step. For initial experimentation and tuning, we use a fixed split of 7 : 3 between training and test sets. The final results shown in this work, however, were obtained using 10-fold cross validation (using a split of 9 : 1). The results of the 10 cross validation runs are averaged, with their standard deviation provided together with the arithmetic mean. This is a widely accepted standard procedure in the evaluation of prediction solutions [161]. We use precision, recall, and the MCC, as described in Section 2.3, to measure the performance of our solution.

Functional Evaluation

Our first experiment aims at verifying the overall performance of our approach. For this, we use the real-world dataset described in Section 4.4.1 as input for our EFP component. The resulting confusion matrix is shown in Table 4.5 (all standard deviations σ are below 0.4 and not shown in the table), and the mean values of the metrics are summarized in Table 4.6.

The results show a precision of 0.873, a recall of 0.971 and an MCC of 0.879. The classifier resulting from the ML model training performs steadily across the entire real-world dataset. It is noteworthy that the standard deviation is relatively low. This, together with the cross validation used, shows that the performance is not dependent on which one of the partitions was used for training, and which one was used for testing.

Impact of Global and Local Events

In the second part of our experimentation, we seek to determine the impact of the available data. In other words, we are interested in how much the visibility of private events impacts the prediction performance. This requires using the

synthetic dataset with its inter-organizational events. Since within this dataset, we inject failures with a given rate, we first examine the impact of this injection rate parameter on the results. We analyze the sensitivity to this parameter in order to avoid bias stemming from parameter choice.

We show the results for precision, recall, and MCC in four different scenarios. In the *global* scenario, events from all process partners are available to the EFP component. This corresponds to having all events marked as public events. In contrast, the *local* scenario only provides the EFP component with events from a single partner. This represents the privacy scenario from the use case described in Section 4.4.1. From these two scenarios, we derive two further scenarios, leaving out the context events, providing a baseline to compare the results to. All values are provided together with their standard deviations marked using error bars. The results for precision are shown in Figure 4.7, recall is shown in Figure 4.8, and the results for MCC are shown in Figure 4.9. The intuition that less available data decreases classification performance is clearly visible. Nevertheless, the data gives insights into the amount of performance decrease caused by only using the data from a specific partner in the evaluation, and also provides a comparison of results between scenarios with context (*global* and *local*) and without context events (*no context*). The biggest drop in performance is seen for recall at a fault rate of 0.10, where the *local* scenario reduces the recall value from 0.972 to 0.299. Removing context events generally decreases precision and recall and therefore also the MCC, except for the corner cases of very low fault rates (where both *no context* scenarios have better recall values than *local*), and very high fault rates (where both *no context* scenarios have better precision values than *local*). In any case, the *global* scenario shows significantly better results than *local* and *no context* across the entire evaluation domain.

Furthermore, analyzing the impact of varying fault injection rates in the synthetic dataset, we see that while precision and recall generally increase with a higher amount of faults, the MCC shows a *sweet spot* around 0.50, for all three executed scenarios. This knowledge has no universal application, since the fault rate is highly domain-specific. Comparing this data to the real-world dataset, however, shows that results when using the real-world fault rate (35%) are comparable to the *global* scenario.

Discussion and Limitations

Generally, we observe an increase in both precision and recall for an increasing fault rate. While it is difficult to determine a reason for certain performance metrics, we suspect that the types of processes used in the dataset cause a high failure rate to be easier for ANNs to process than low failure rates. The *global* scenario generally shows better recall than precision. In contrast, the *local* scenario shows higher precision values than recall, especially for low fault rates. We suspect that the inter-organizational structure of the process involved in the synthetic dataset benefits an accurate detection of inherent faults, increasing the precision. This, in turn, comes at the cost of lower recall.

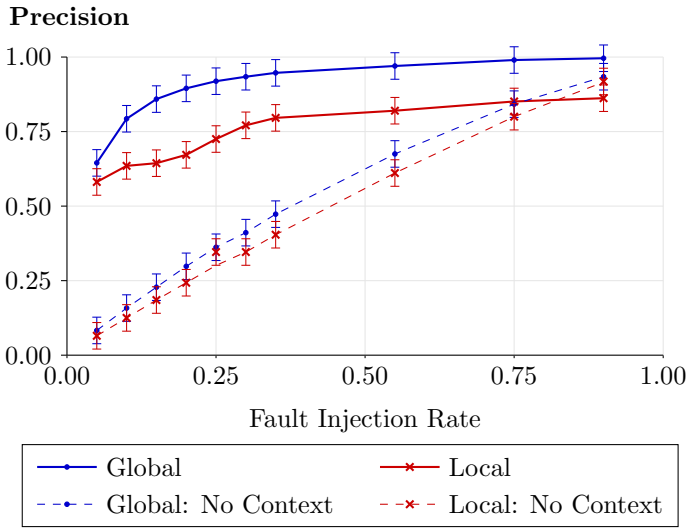

Figure 4.7: Precision over fault rates for synthetic dataset

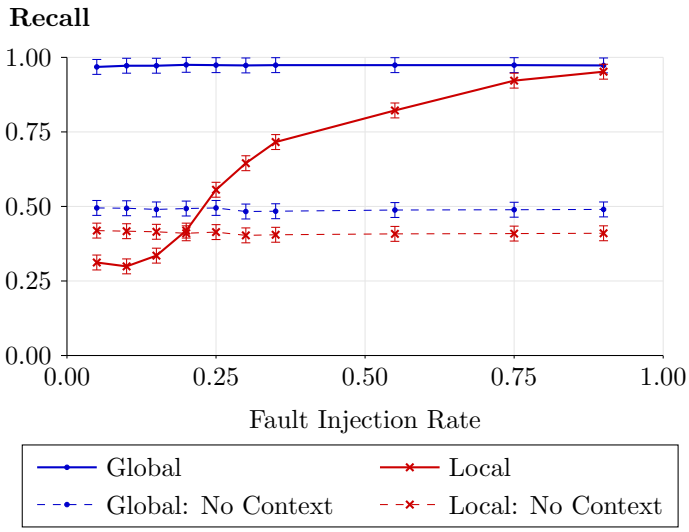

Figure 4.8: Recall over fault rates for synthetic dataset

The *no context* scenarios aim at giving a baseline for our approach by not taking into account context events at all. We can observe this scenario to be superior to *local* for high fault rates (> 0.75) with regard to precision, and for low fault rates (< 0.20) with regard to recall. These figures represent some noise that context events can add, distorting the prediction for those extreme cases of fault rates. The MCC, which aims to find a one-metric balance between many possible metrics to compare classifiers, however, is consistently higher for *local* than for *no context*, and the highest for *global*. From these results, we conclude that

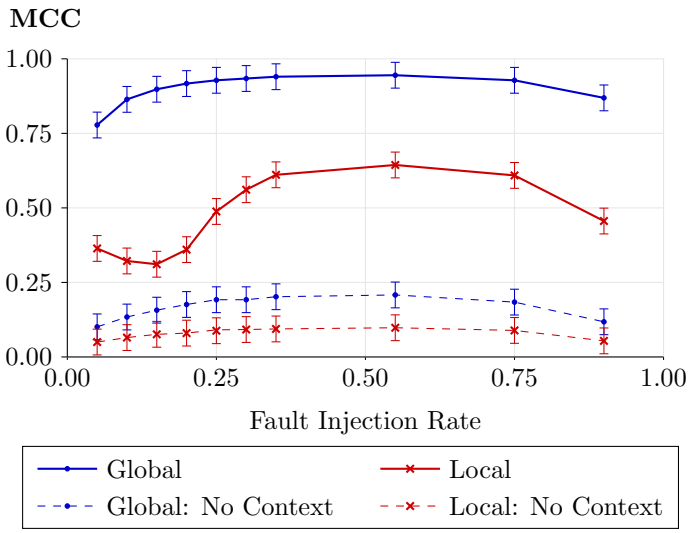

Figure 4.9: MCC over fault rates for synthetic dataset

considering context events clearly yields significant improvements in precision, recall and MCC values.

Finally, we note that while our approach does not require experts to create rule-based failure prediction models, since ML is used, it still requires certain expert input: (i) The creation of a suitable ML model still requires expert decisions for selecting ML models and tuning parameters, and (ii) the inclusion of certain data sources as external events requires domain-specific knowledge. However, we argue that the area of expertise required is different. While for creating a rule-based prediction model, experts in the given domain are necessary, in our case, the required knowledge is more abstract, as skills in ML are required, and domain-specific knowledge is less relevant.

Result Summary

The proposed approach yields satisfactory results for predicting failures in business processes. The resulting metrics show a precision of 0.873 ($\sigma = 0.014$), a recall of 0.971 ($\sigma = 0.008$) and an MCC of 0.879 ($\sigma = 0.012$). In the second part of our evaluation, we analyze the performance when facing a scenario where multiple process partners collaborate, but do not share all of their events, which is possible in situations where privacy aspects prevent a business process partner from sharing internal events. We observe that the impact on the performance is measurable, and, depending on the metric and fault rate, can decrease the prediction performance from around 0.972 to 0.299 at a fault rate of 0.10.

It is noteworthy that this is a parameterizable approach, and the best learning model may differ from domain to domain. Therefore, while our approach reduces the necessary expert knowledge required to create failure prediction models, there is still the need for an expert for parameter tuning to achieve satisfactory results.

Nevertheless, our approach can provide a flexible and generic methodology of predicting failures in business process execution.

4.5 Related Work

The usage of events in BPM has gained attention in recent years, with a focus on fields like Complex Event Processing (CEP) for business processes, Business Process Intelligence (BPI), and Business Activity Monitoring (BAM) [142, 164]. In addition, work has been done in the area of failure prediction and fault tolerance, as discussed in more detail below. A more extensive discussion of event-based BPM can be found in [164, 232]. Nonetheless, only few approaches consider the execution context of BPM, thus exploiting the presence of context events in combination with internal events, i.e., those generated internally by the process execution.

Integration of BPMS and EBS

Event-based BPM relies on the principle of Event-Driven Architecture (EDA), which describes an architectural style with event-driven components and communication [179]. EDA resemble technical aspects of publish/subscribe middleware [88], especially regarding the decoupling of components, and the pushing of events [44]. In event-based BPM, an EDA allows to communicate events between different process stakeholders, components, and process steps, and therefore to control and change business processes during runtime and design time [6]. In the process of distributing the execution of BPMS [174], several works exploit the decoupling properties of publish/subscribe systems, to allow the distributed execution of business processes [144, 223]. The idea is to exploit the loosely coupled and distributed nature of publish/subscribe systems. The BPMS becomes an event sink as well as an event source, thus generating and consuming process-related events [87, 232].

Importantly, the BPMS can be controlled by an EBS through events [142]. In an early approach to use events in BPM, von Ammon et al. [7] present a basic reference model to control processes. Event types from Business Process Model and Notation (BPMN) 1.1 are supported, which includes exception events. The main focus of this work is on a generic approach to use events to control a process instance. Hence, the authors do not discuss how exception events could be generated or how to derive failures from events.

An example for the usage of CEP in order to adapt a business process instance is presented by Hermosillo et al. [124]. The authors propose to adapt a process at runtime based on predefined, event-based rules, which makes the approach rather inflexible. The main contribution of this work is a language to define these rules and when and how to apply them; nevertheless, there is no discussion on the nature of the events and how to identify a critical event if it has not been specified in a rule. A related approach has been presented for scientific workflows [276]. Here, event messages are emitted by distributed agents for

workflow execution control. Only events explicitly generated by agents are taken into account, i.e., context events are not explicitly regarded. Reactions to events are done based on predefined rules.

Several authors have proposed event-driven monitoring approaches. For instance, Feldman et al. [97] show an effective example of event-based prediction. In their use case, real-time monitoring data is used to anticipate issues of cargo shipments. The prediction model relies on a simple stochastic approach, therefore, it is able to discover only direct relations between system state and predicted outcome. More sophisticated approaches to prediction can be investigated, as is shown by Schwegmann et al. [233]. The authors combine real-time event monitoring with predictions of future process behavior. The resulting model allows the usage of a number of ML-based predictors, similarly to our approach. In contrast to our work, this approach focuses on the prediction of categories and numbers, not on failures, and only takes into account events generated by a BPMS.

Our work combines a BPMS with an EBS aiming to monitor the execution of BPM and to perform predictive analysis. We are not interested in discovering complex events, but rather in processing the huge amount of BPM-related data to predict the future system evolution. Indeed, similarly to the work by Schwegmann et al. [233], we rely on a ML technique for predicting failures. However, in contrast to all previously discussed work, we augment the events generated by the BPMS running the business processes with context-related events; the latter can encompass events generated by IoT devices (e.g., temperature or position sensors) placed in the real-world execution environment of the business process. By distinguishing between intrinsic and context events, we strengthen separation of concerns. This, in turn, allows for a better reuse of the shared context among multiple business processes.

Failure Prediction

Apart from the approaches discussed so far, which focus on generic process adaptation and BAM, there is also a number of approaches explicitly aiming at fault tolerance for business process executions. From a technical point of view, different strategies to make service compositions fault-tolerant have been proposed [277]. Fan et al. [91] introduce a fault tolerance strategy for service compositions which includes failure detection. However, the detection is done by comparing an expected result with the actual outcome of a service composition. Hence, the approach is rather inflexible and requires modeling of the expected results; an actual failure prediction is not carried out. Events are generally not taken into account, only the outcome of a service composition is assessed. In addition, a large number of approaches to tolerating *non-functional* faults (e.g., delays) in service compositions have been proposed. For instance, Leitner et al. [170] use ML techniques to predict performance faults. In an earlier approach, Canfora et al. [48] propose to re-plan service compositions at runtime based on QoS. The approaches discussed in this paragraph focus on the technical level of service compositions, not taking into account context events

as observed in the work at hand, but rather aiming at failures arising from the execution of software-based process steps.

Failure prediction for business processes is also partially related to the field of anomaly detection in process executions. For instance, Bezerra et al. [23] use process mining in order to classify anomalous and normal instances of a particular process model. The outcome of this anomaly detection is an *ex post* analysis whether something uncommon occurred. Also, the approach does not consider events; instead, process logs are mined. Hence, only anomalous process steps are taken into account, while we argue that context events actually may precede such steps at runtime.

To grasp the complex relation among system components and to discover early symptoms of failures to come, several approaches in existing literature rely on ML techniques. Abu-Samah et al. [3] rely on Bayesian networks; as a drawback, this approach requires to be complemented with the extraction and validation of system patterns, which may involve expert opinions or elicitations on several levels. Leontjeva et al. [173] address the problem of predicting the (positive or negative) outcome of an ongoing business process by analyzing event logs using a Hidden Markov Model. As such, the solution assumes the Markovian property, therefore it cannot easily take into account long-term dependencies among events and outcome, like we do. An approach based on Hidden Semi-Markov Models, which loosen the Markovian property, has been presented in [224]. Pika et al. [211] provide a solution based on statistical analysis aimed to identify the risk of deadline overruns of processes. A more flexible solution is proposed in [152], where Kang et al. aim at the detection of abnormal process *termination*, and, similarly to our work, the authors apply ML to achieve real-time fault detection. However, the authors focus on process-intrinsic knowledge, while context events are not taken into account. In the end, their approach compares the actual process execution with the expected process execution, again requiring expert knowledge (definition of the expected process execution). Nevertheless, this work comes closest to the work at hand. Although focused on process events only, an interesting approach is proposed by Teinemaa et al. [250]. It jointly exploits unstructured (free-text) and structured data to predict the process outcome. Even though we do not consider unstructured data, our approach could embed the principles presented in [250].

Grambow et al. [112] provide an approach to event-based exception handling for processes. Importantly, the authors focus on software engineering processes, not on business processes in general. Their approach to identifying critical events is based on the event-condition-action pattern, which makes it necessary to define events and conditions to handle them. While the authors claim that their approach is able to take into account unanticipated conditions, it remains unclear how this is achieved. Events are related to process activities and artifacts, while context event sources are not regarded. In a more specialized

approach, Pika et al. [212] aim at process risk management through the analysis of event logs. Since the authors focus on risk management aspects, their work exceeds the work at hand in terms of prediction of a process outcome: While we focus on failure prediction, Pika et al. also predict predefined possible process outcomes, e.g., timeliness of single process steps. To identify risks, process models are annotated with guards, while in our approach, we do not require such prearrangements.

Other works focus on identifying root causes of failures. Conforti et al. [66] propose another approach aiming at process risk management based on the analysis of event logs. The goal of the authors is to minimize risks by identifying potential risks (e.g., timeliness, reputation, cost) during the scheduling of work items. A ML approach is applied on process-related events, thus neglecting context event sources. Examples for root cause analysis based on event data and using ML have been proposed before [221, 244]. While we do not focus on root cause analysis for process faults, this could be another interesting direction for future work.

Context Awareness

Context awareness helps to improve decision-making processes by introducing new information that can better describe the execution conditions of a business process. Despite the importance of context information, only few works consider them explicitly. In [249], the authors investigate the most commonly used kind of context information employed so far (but not in the field of BPM). The authors conclude by assessing that performing adaptation in response to changing execution condition, captured by context information, may be very beneficial to software systems. A similar idea results from the work in [124], which relies on events to control the BPM process execution. Nevertheless, the latter proposes a rule-based approach which may not be flexible enough with regard to previously unknown context information. Conversely, we leverage on the flexibility of ML techniques to work in presence of concept drifts [264].

Böhmer and Rinderle-Ma [28] provide a runtime approach to anomaly detection which also takes into account the context of a process, including time- and resource-related events. The main goal is to identify malicious attacks on process executions. The authors base their approach on an explicit set of expected execution events and their likelihoods of occurrence. In contrast, our approach only considers the expected execution events in an implicit way, thus leading to higher flexibility. Also, the approach presented by the authors focuses on a predefined set of event sources, while we allow the integration of arbitrary sources, through the concept of context events.

Related Work Summary

Our presented approach is novel since it proposes an integrated solution which takes into account the following key points which have not yet been fully regarded in the existing literature. First, most related work focuses on event logs, not foreseeing adding arbitrary events, e.g., from the IoT. Through integrating external data sources and XES as a common data format, we are able to achieve

this. Second, the current state of the art in failure prediction mostly depends on predefined rules or conditions when a particular failure will arise. Through the application of a ML-based approach, our solution is more flexible, though it still needs labeled training data. Third, we do not restrict our approach to a particular goal, such as monitoring, process adaptation, or risk management, but aim at generic failure prediction. Fourth, to the best of our knowledge, there is currently no discussion on how the visibility of event data in inter-organizational settings influences predictions of process outcomes. We conduct such an analysis as part of our evaluation in Section 4.4.

4.6 Summary

We have discussed the need for methodologies to predict and respond to unforeseen events and failures, and presented an approach to employing event-based error detection and failure prediction for business processes. We have evaluated our approach using two datasets. The first dataset is a real-world business process dataset from the financial domain. The second dataset is a synthetic dataset modeled after a realistic scenario, stemming from a choreography model involving several partners collaborating in a common business process.

We demonstrate that the implemented failure prediction component is indeed capable of detecting future failures in business processes. Our evaluation not only shows that our solution is suitable for predicting failures accurately—in the real-world dataset experiments, the failure prediction exhibits a precision of 0.873, a recall of 0.971 and an MCC of 0.879—but also provides insight into the impact of private and public events for varying fault rates.

CHAPTER 5

Predictive Cloud Scaling

In Section 2.1, we discuss that elasticity is a core property of contemporary distributed systems. By extension, this also applies to DSP [11, 121]. Using the scalability of the underlying infrastructure, Stream Processing Engines (SPEs) dynamically adapt to changes in the input data rate during runtime, reducing cost while aiming at meeting a predefined QoS level [114].

As with all types of distributed systems, in DSP, every scaling operation requires resources by itself. In the case of DSP, scaling incurs a delay, during which the DSP system must wait for the resource (e.g., a VM) to become available, and it consumes energy, leading to computational overhead [108, 185], and therefore increased cost. Therefore, while being crucial for elasticity, intensive operations such as scaling and migration should be kept at a minimum [68, 72, 130, 185].

Most approaches to scaling in DSP use thresholds for resource utilization [68, 131, 185, 189], as discussed in Section 2.2. For instance, a DSP system using threshold-based scaling tries to maintain its load within the defined thresholds by scaling out (activating new DSP operators) or scaling in (passivating unused DSP operators) [11, 114]. However, such an approach can result in frequent scaling operations, incurring an overhead of resource usage and cost [68]. In certain cases, this overhead is necessary in order to benefit from the additional computing power, to avoid under-provisioning, or to save energy. However, excessive scaling also increases the risk of unnecessary cost [68, 185].

System metrics used for reaching scaling decisions can be considered as time series, containing both long-term trends in data rate, as well as short-term variances (spikes and valleys) [178]. The long-term trend can be the development of input data depending, for instance, on the time of day (e.g., peak hours) or time of year (e.g., summer holidays), while short-term spikes might stem from spontaneous and short-lived events like bursts in network communication.

The latter represent noise that we aim to ignore for scaling decisions to avoid unnecessary cost [68].

In this chapter, we propose to extend classic threshold-based scaling in DSP by improving the scaling mechanism's reaction to load changes. Instead of relying on one metric and employing simple threshold-based scaling, we observe both intrinsic parameters of a SPE, as well as extrinsic metrics. From these metrics, we derive an estimated *true* inner state, neglecting noise (i.e., the short-term variance) by separating it from the long-term trend. Based on this estimated state, more stable and robust scaling decisions can be reached. We propose a concrete scaling mechanism applying EKFs [143, 151] to perform this state estimation, and evaluate our approach in detail using a testbed with an image processing workload.

In the following, we introduce a motivational use case scenario for DSP in order to exemplify the usage of DSP in an area with real-world applicability, modeled after a real-world scenario [137].

> **Example Scenario.** A Minnesota-based research group in the field of biomedical engineering is capturing microscopic images of biological cell tissue, which are processed in various ways in order to gain insight into a given tissue sample. The images are captured pairwise in a tile-like fashion. Figure 5.1 shows a sample pair of images captured using two different illumination wavelengths (left, center). An image resulting from an overlay of those individual images, as used by the biomedical research group, is shown on the right. The images are then passed through a composition of DSP operators, called a topology. Within this DSP topology, the image pairs are processed using various operators (thresholds, pixel value mapping, noise reduction, etc.) and the results are used to create statistical data about cell cultures. This statistical data is then used by the researchers to determine aspects and features of a tissue sample. This processing is performed on an elastic DSP platform, where VMs are used to host DSP operators.
>
> Instead of reacting to every spike and valley of its load measurements, the DSP platform uses EKF-based filtering to achieve a smooth time series of measurements. This way, it can react quickly to actual changes in load without wrongly reacting to measurement noise.

5.1 Scaling using Extended Kalman Filters

The goal of our work is to minimize the amount of scaling operations, while maintaining rapid elasticity, to avoid the cost of overly frequent scaling [68, 108, 185]. We regard each operator within a DSP separately, and measure the amount of incoming data (*data rate*) and the system state, with the goal of

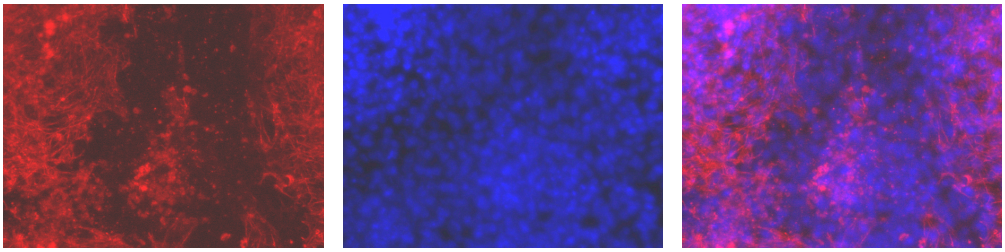

Figure 5.1: Sample images from the example scenario

reaching a scaling decision. This decision can be to either (a) scale out by starting more operator instances, (b) to scale in by shutting down instances, or (c) to remain in the same state. Each operator is executed using one or more operator instances, determined by the scaling decision. We denote the set of operator instances for any given operator as B, with instances $b_1, b_2, b_3, \ldots, b_n$, where $n = |B|$. Table 5.1 gives an overview of the notation used in this chapter.

In this scenario, the scaling decision is reached based on both intrinsic metrics (the system state) and extrinsic metrics (the amount of incoming data). Together, these two data sources form a multivariate dynamic system, which can very well be modeled using EKFs, since EKFs provide a vector-based state and transition representation. In EKFs, multiple state variables (metrics) can be included in the system state vector. Possible metrics include the system's CPU load [114], its memory utilization [68], network traffic [274], throughput and queue sizes [98], or other metrics determining the system's overall performance [45].

5.1.1 Control Loop

System, Feedback, Environment, Controller

We treat the observed operator, the rate of incoming data, and the scaling mechanism which controls the scaling decisions of the operator, as a control system. Our approach involves creating a *closed control loop* [109], a commonly used model for scaling [178], shown in Figure 5.2. During operation, the *system* (the stream processing operator) is under constant supply of input data, measured by its rate. Since the amount of incoming data is outside of our control, we define this as the operator's *environment*[1]. The stream processing operator is controlled by the scaling mechanism, being the *controller* in our control loop. The controller is responsible for reaching scaling decisions, i.e., defining whether the operator must be scaled out, scaled in, or can remain unchanged. During the processing of data, the operator is influencing the system state (e.g., through CPU load or memory utilization), constituting the *feedback*, which is measured by the controller. The controller therefore has two sources of

[1]Known as *disturbance* in other literature [151].

Table 5.1: EKF notation

Model	
B	Set of operator instances
$b_1, b_2, \ldots$	Individual operator instances in B
Θ^-	Scale-in threshold
Θ^+	Scale-out threshold
Time Series and Filtering	
T	Set of all measurement times
$t_1, t_2, \ldots$	Individual timestamps in T
Z	Time series of system state measurements
$Z_{t_1}, Z_{t_2}, \ldots$	Individual state measurements at time t in Z
$\lambda(\cdot)$	Filtering (smoothing) function
$\lambda_Z(t)$	λ at time t based on history Z
Z'	Filtered system state time series
$Z'_{t_1}, Z'_{t_2}, \ldots$	Individual filtered measurements at time t in Z'
Measurements and EKF Model	
D_t	Data rate at time t
ΔD_t	Date rate change from $t-1$ to t
$u_t = (D_t, \Delta D_t)$	System input at time t
x_t	System state at time t
$\hat{x}_t^*$	System state estimation for time t, a priori
$\hat{x}_t$	System state estimation for time t, a posteriori
z_t	Measurement (observation) at time t
$\hat{z}_t$	Measurement estimation for time t
w_t	System noise at time t
v_t	Measurement noise at time t
Q	System noise covariance
R	Measurement noise covariance
$f(x_t, u_t)$	State transition function
$h(x_t)$	Measurement function (in our case $h(x) = x$)
EKF-Internal State Matrices	
P_t	Estimation error at time t
F_t	Jacobian matrix of $f(\hat{x}_t, u_t)$
H_t	Jacobian matrix of $h(\hat{x}_t)$
G_t	Kalman gain at time t
Miscellaneous	
$\mathcal{N}(\mu, \sigma^2)$	Normal distribution, variance σ^2 around μ
X^T	Transpose of matrix X
X^{-1}	Inverse of matrix X

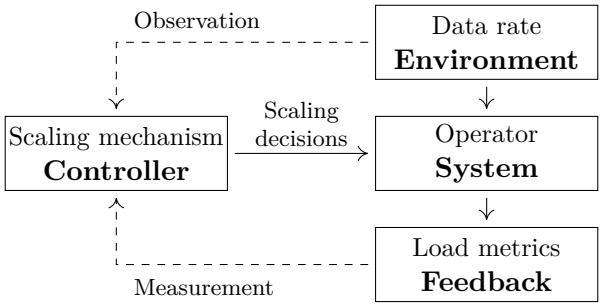

Figure 5.2: Overview of the proposed approach, modeled as a control loop

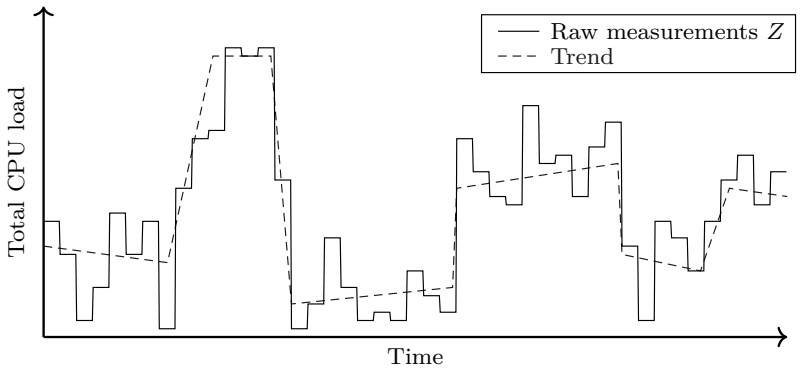

Figure 5.3: Example of long-term load trend (dashed) and measurements (solid)

information for reaching scaling decisions: the rate of incoming data, and the system state of the operator.

5.1.2 Filtering Model

As discussed above, we measure the system state over time, and base our scaling decisions on the measured values. However, we employ a filtering of the raw measurements to create a smoother version of the measurement curve.

The time series of recorded measurements of the system state is denoted as Z. In practice, those measurements are offset from a trend by a certain noise. This noise can have numerous causes, ranging from disturbances at the OS level, hypervisor strategies at the VM level, or workload shared with other applications.

All of these aspects cause the system state (e.g., CPU load or memory utilization) to exhibit high variance. Nevertheless, a certain trend is always present, e.g., a highly demanding processing node will have a given baseline (trend) of load throughout its operation. Figure 5.3 illustrates such a scenario, using the CPU load as an example of fluctuating system state.

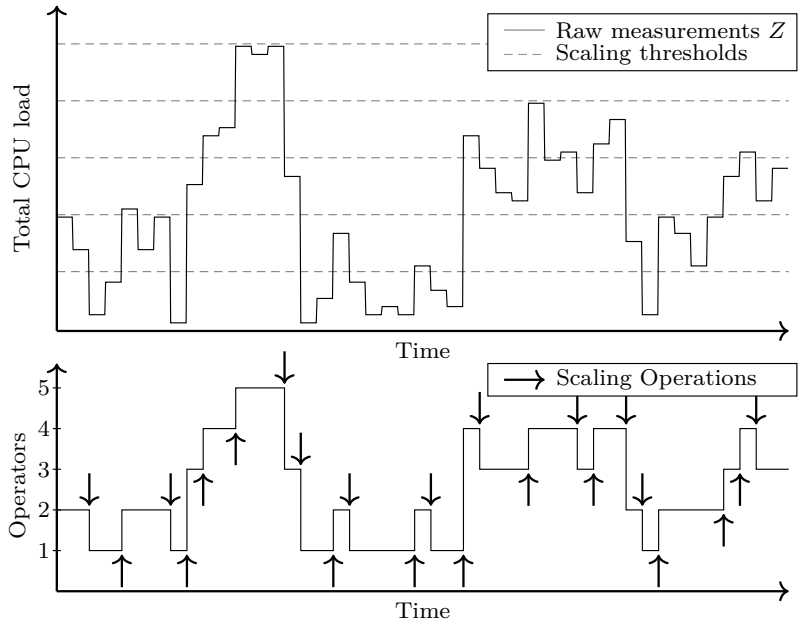

Figure 5.4: Scaling of operators according to thresholds of the actual load

If a stream processing system bases scaling decisions purely on the raw data, an excessive amount of scaling operations may occur [68, 108, 185]. This is illustrated in Figure 5.4, where the fluctuating CPU load measurements (top graph) lead to 23 scaling operations, i.e., excessive scaling (bottom graph). Our approach applies filters to this process to reduce the number of scaling operations, i.e., to reduce the number of steps in the *operators* line in Figure 5.4.

Therefore, we formally define our approach as follows. We regard a history (time series) of raw measurements, Z, at various points in time t out of all measured times T, where Z_t is the measured state at time t, as shown in (5.2).

$$T = \{t_0, t_1, \ldots, t_n\} \tag{5.1}$$

$$Z = \bigcup_{t \in T} Z_t = \{Z_{t_0}, Z_{t_1}, \ldots, Z_{t_n}\} \tag{5.2}$$

Based on the raw measurements $Z_t \in Z$, we employ a filter, which we denote as $\lambda(\cdot)$, and apply this filter to each Z_t, i.e., we calculate $\lambda(Z_t)$. This application is performed at every given measurement time t and has access to all other measurement values in Z, with the practical limitation that it can only access past measurements. We therefore define $\lambda_Z(t)$ as the filtered value of Z at time t, given all other values $Z_i \in Z$ where $i \leq t$. For λ, various filters can be used. In our work, we use an EKF as a smoothing filter, which we will describe in

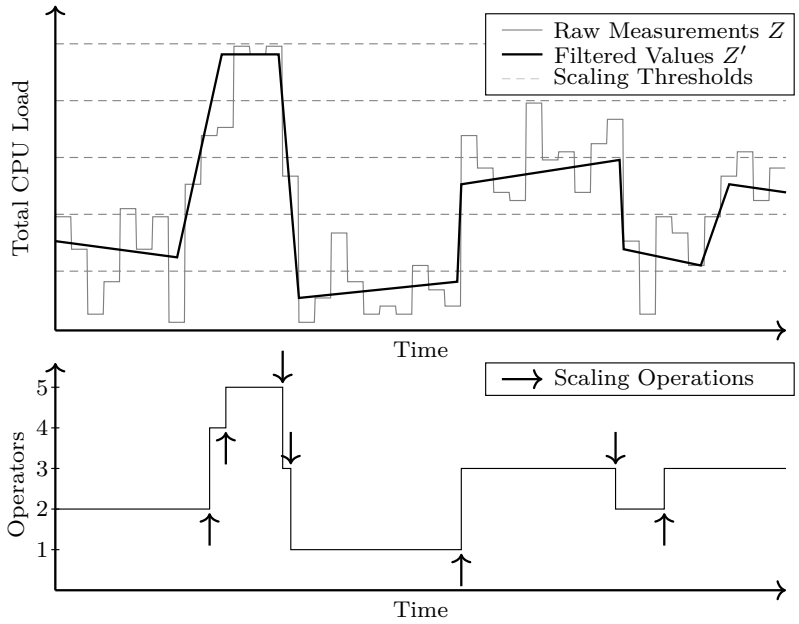

Figure 5.5: The same scenario, with additional measurement filtering

Section 5.1.3. Other filters like Linear Smoothing (LS) [236], TVD [235], or simpler versions of the EKF [33] are used for smoothing in literature, but in most cases not with regard to DSP. Related work is discussed in detail in Section 5.4.

We define the set of filtered measurements Z' as shown in (5.4).

$$\forall Z_t \in Z : Z'_t = \lambda_Z(t) \tag{5.3}$$

$$Z' = \bigcup_{t \in T} Z'_t = \{Z'_{t_0}, Z'_{t_1}, \ldots, Z'_{t_n}\} \tag{5.4}$$

Figure 5.5 shows a possible resulting graph of the same data rate measurements as shown in Figure 5.4, using a filter, along with the resulting scaling behavior of the system. When compared to Figure 5.4, it becomes clear that the amount of scaling operations has decreased, and only 7 scaling operations remain.

5.1.3 Extended Kalman Filters

In the following, we describe EKFs, the type of underlying filter used in our approach, along with the concrete state transition model.

The EKF [143] is a nonlinear generalization of the Kalman Filter (KF) [151]. Kalman-type filters work by defining models for state transitions of the system, as well as models for the observations (measurements) of the system. While regular

KFs use purely linear transition models, i.e., matrices and linear algebra, EKFs generalize the algorithm for nonlinear models. Instead of matrices, EKFs use functions as transition models, and require both the transition and observation function to be point-wise differentiable.

Note that the EKF is not the only suitable model usable for forecasting multivariate processes. Especially in processes with a high potential for repeating patterns, autoregressive models such as Autoregressive Moving Average (ARMA) or Autoregressive Integrated Moving Average (ARIMA) are used [117, 253]. However, the unique advantage of EKF-based filters over models such as ARMA or ARIMA is that with some—even inaccurate—knowledge of the underlying system model, not only the measurements of system state are incorporated into the solution, but also the (very accurately known) system input. At the same time, the EKF maintains covariance matrices determining the current confidence into each data source (both system state and input).

System

The first steps for defining our EKF are as follows: The *system state* is denoted as x. This vector can include multiple state variables (metrics), such as CPU and memory utilization, network traffic, throughput or queue sizes. Since the state changes over time, we use x_t to indicate the state at time t. Furthermore, our system is controlled by an external input, which is the rate of data sent to the stream processing operator. We observe both the momentary data rate at time t, defined as D_t, as well as the change in data rate compared to the previous value, ΔD_t, with $\Delta D_t = D_t - D_{t-1}$. Together, we define the *input* to the stream processing operator at a given time t as $u_t = (D_t, \Delta D_t)$. Since our operator is running on a real-world computer, and therefore is subject to fluctuations in performance, the state also encounters a particular *noise*. We denote this system noise as w_t for a given time t. Finally, we define our *state transition model*, which models the system state x_t, based on the previous system state x_{t-1}, the input u_{t-1}, and the system noise w_t, as shown in (5.5), where $f(\cdot)$ represents the *state transition function*, which depends on the last state x_t and the system input u_t. The system noise w_t, according to the original definition of KFs [151], is assumed to be zero-mean Gaussian noise[2] with the covariance matrix Q, as shown in (5.6).

$$x_t = f(x_{t-1}, u_{t-1}) + w_t \tag{5.5}$$

$$w_t \sim \mathcal{N}(0, Q) \tag{5.6}$$

The state transition function $f(x, u)$ can be chosen independently of the remaining part of this state system. In our scenario, for simplicity, we use a linear state transition function, based on the current system state x, and the input

[2] We discuss this assumption of zero-mean Gaussian noise for our scenario in Section 5.2.3.

$u = (D, \Delta D)$, defined as $f(x, u) = x + a \cdot D + b \cdot \Delta D$. However, due to the usage of EKF, a nonlinear state transition function could also be used if nonlinear dynamics are known. The vector parameters a and b define the sensitivity of the EKF to the input data rate, and must be defined according to the workload. Currently, these parameters are determined using Ordinary Least Squares (OLS) linear regression, but in future work, this technique can be extended to use ML.

Feedback

Next, we take into account the measurement of the system state, i.e., the feedback. There are several mechanisms for measuring system load, e.g., stand-alone programs like `top` and `ps`, or APIs for direct measurements. The EKF definition includes a measurement function, which we denote as $h(\cdot)$. This function takes the system state x and transforms it into a measured value. This value is again subject to noise, this time, stemming from the measurement process itself. According to EKF terminology, we call this the *measurement noise* and denote it as v_t. For physical sensors, this represents a measurement error or inaccuracy. In our scenario, this measurement error represents the inaccuracy of measuring CPU load. The resulting measurement is denoted as z_t, and defined (5.7) where v_t, the measurement noise, is again assumed to be zero-mean Gaussian noise, and its covariance matrix is assumed to be R, as shown in (5.8).

$$z_t = h(x_t) + v_t \tag{5.7}$$

$$v_t \sim \mathcal{N}(0, R) \tag{5.8}$$

In contrast to physical sensors like temperature or light sensors, which often have nonlinear characteristics, or require unit conversion, we do not need transformations in the process of measurement. Therefore, we simply define $h(x_t) = x_t$, and the resulting measurement is reduced to the term shown in (5.9).

$$z_t = x_t + v_t \tag{5.9}$$

Figure 5.6 gives an overview of the dynamics of the described state system. While the input to the system (u_t) is known but not controllable, the system's true state is hidden from the controller. This includes the noise influencing the system state itself (w_t), as well as the noise of the measurements (v_t). Only the result of the measurements (z_t) is visible to the controller. As described before, $u_t = (D_t, \Delta D_t)$, f is the state transition function, h is the identity function $h(x_t) = x_t$, $w_t \sim \mathcal{N}(0, Q)$, and $v_t \sim \mathcal{N}(0, R)$.

Note that the general definition of EKFs allows both $f(\cdot)$ and Q, as well as $h(\cdot)$, and R to be dependent on the time t, i.e., the notations $f_t(\cdot)$, Q_t, $h_t(\cdot)$, and R_t

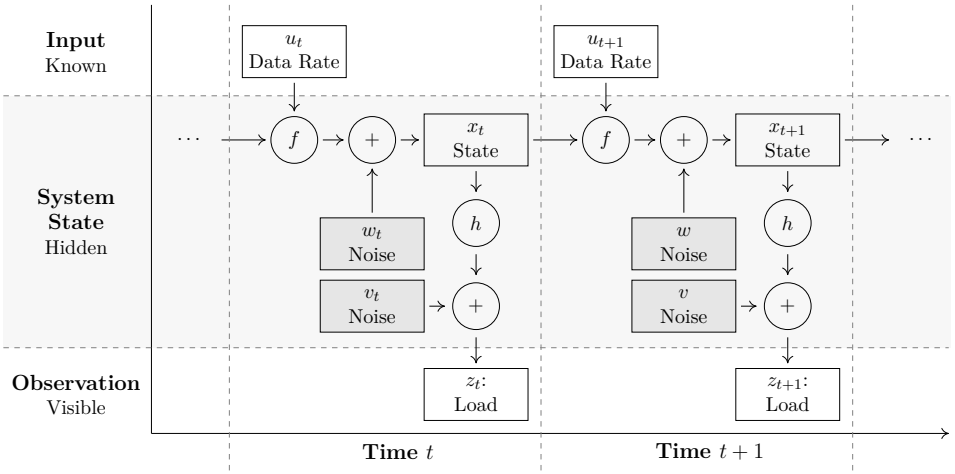

Figure 5.6: State transition system used as a base model

are used, respectively. Since we use time-constant definitions for $f(\cdot)$, Q, $h(\cdot)$ and R in our approach, we drop the index t.

Therefore, our controller reaches scaling decisions based on the rate of incoming data (u) and the (noisy) measurement of system state (z_t). As stated in Section 5.1.2, we do not directly use the measurement z_t, but instead, use the estimation feature of the employed EKF, which yields an estimated version of the *next* system state, denoted as $\hat{x}_t$.

The nature of EKF is that it performs a continuous loop of *predict-update* iterations. Given a current state, in the *predict* step, the EKF performs a prediction of the next system state. In addition, the EKF also provides the *prediction error*, which can be used as a metric of confidence in the predicted value. Then, provided with a (noisy) measurement of the actual value, the EKF recalculates its prediction error, and provides a new prediction in the *update* step. As a result, the EKF is constantly correcting its prediction, provided with noisy measurements, while maintaining a balance between inaccuracy in measurements, as well as external disturbances. Furthermore, this process takes into account the *input* of the system, i.e., control variables which are manipulated externally. In our scenario, this is the amount of data rate and its change, i.e., $u_t = (D_t, \Delta D_t)$.

Predict Step

At any point in time t, from a given previous estimated system state vector $\hat{x}_{t-1}$—either the initial state (see Section 5.1.4 for a description of bootstrapping in our approach) or the previously estimated state—and the previous system input u_{t-1}, the EKF derives both the estimated *a priori* next system state $\hat{x}_t^*$, the estimated next measurement $\hat{z}_t$, and the estimation error P_t, as shown in (5.10)–(5.12), respectively. Here, F_{t-1} is the Jacobian matrix of f, i.e., the matrix of partial derivatives of the state transition function for the current state

and input $f'(\hat{x}_{t-1}, u_{t-1})$, P_t is the prediction error (covariance matrix) at time t, and Q is the covariance of the system noise as defined above. P_t is computed by applying the Jacobian matrix of the state transition matrix F to the previous prediction error, then re-applying its transpose F^T, and finally adding the system noise covariance Q. This prediction error will later be used to calculate the *Kalman gain* G_t. Its definition is derived from the EKF proposal [143]. The initialization value for P_0 is discussed in Section 5.1.4.

$$\hat{x}_t^* = f(\hat{x}_{t-1}, u_{t-1}) \tag{5.10}$$
$$\hat{z}_t = h(\hat{x}_t^*) \tag{5.11}$$
$$P_t = F_{t-1}\, P_{t-1}\, F_{t-1}^T + Q \tag{5.12}$$

Update Step

After a new measurement z_t is taken, the EKF updates its matrices and vectors to reflect the new data. First, the Kalman gain G_t is calculated, which is used to create the new *a posteriori* system state estimate $\hat{x}_t$. Note that the difference between the *a priori* estimate $\hat{x}_t^*$ and the *a posteriori* state estimate $\hat{x}_t$ is that the *a posteriori* state incorporates the new measurement (and therefore, new knowledge) into the value provided by the *a priori* state before the measurement.

The update step for the Kalman gain G_t and the estimation of the next system state $\hat{x}_t$ is shown in (5.13) and (5.14), respectively, where H is the Jacobian matrix of h, i.e., the matrix of partial derivatives of the measurement function for the current measurement $h'(\hat{x}_t)$, and R is the covariance of the measurement noise as defined above. Again, this definition is derived from the original work in [143]. The Kalman gain G_t is used in the estimation of the next system state $\hat{x}_t$ and represents the (estimated) influence of the change in measurement on the actual system state.

$$G_t = P_t\, H^T (H\, P_t\, H^T + R)^{-1} \tag{5.13}$$
$$\hat{x}_t = \hat{x}_t^* + G_t(z_t - \hat{z}_t) \tag{5.14}$$

Revisiting Section 5.1.2, we are now able to define the filtered version of z_t, i.e., $Z_t' = \lambda_Z(t)$, by using the cumulative output of the EKF estimations for each operator instance, as shown in (5.15), where B, as defined at the beginning of Section 5.1, is the set of all operator instances for the operator type taken into account, $\mathrm{b} \in B$ denotes the iteration over all operator instances, and $\hat{x}_t^b$ denotes the EKF estimation $\hat{x}$ at time t for the operator instance b.

$$Z_t' = \lambda_Z(z_t) = \sum_{\mathrm{b} \in B} \hat{x}_t^b \tag{5.15}$$

5.1.4 Bootstrapping

First, the EKF must be initialized. Since especially at the beginning of the lifetime of an operator, a certain amount of time must be chosen in order for the operator to stabilize, we propose a simple bootstrapping process. In our work, we distinguish between a *cold start* and a *warm start*. If the operator has never been executed before, and therefore its behavior is unknown, a cold start is executed, and a default number of instances is initiated. In our current implementation, this default number is set to one, i.e., if the system has no knowledge about the operator, a single instance of it is spun up. In case the system has already used this operator before, and data about its behavior has been collected, we execute a warm start, and use the average number of instances of the operator used in the last run. This is done to use a value as close to the likely required scale as possible during the bootstrapping process.

After the initiation of the operator instances, we begin a two-step parameter bootstrapping. First, a *dead time* is implemented, during which no scaling decisions are made, and only measurements of the input data rate (u_t) and the system state (z_t) are taken and collected. After the dead time, the EKF is initialized with the following parameters.

Measurement noise covariance matrix R: Since we cannot distinguish between measurement noise and system noise just by measuring the system state (z_t), we use calibration measurements using FakeLoad [237], a dedicated load generator. Given a relatively noiseless load generation, all measured variance represents measurement noise and constitutes R.

Initial state estimation $\hat{x}_0$: To initialize the state estimation, we use a simple weighted average of the measurements of system state (z_t) during the dead time. We weight the system state measurements by recentness, where each weight is indirectly proportional to its age, as shown in (5.16), where n is the number of z measurements, z_i is the i^{th} measurement and Δ_n is the n^{th} triangular number[3].

$$\hat{x}_0 = \sum_{i=1}^{n} \frac{i}{\Delta_n} z_i \tag{5.16}$$

Initial state estimation covariance matrix P_0: This parameter determines the covariance of the prediction, i.e., gives a measurement of the confidence in the estimation of the initial system state $\hat{x}_0$. Since we derive $\hat{x}_0$ from a weighted mean of measurements of z_t during the dead time, we use the

[3]Triangular numbers are defined as $\Delta_n = \sum_{x=1}^{n} x$.

same technique to derive P_0, as shown in (5.17), where $\Delta_n - 1$ represents *Bessel's correction* for an unbiased estimator of covariance [218].

$$P_0 = \sum_{i=1}^{n} \frac{i}{\Delta_n - 1} (z_i - \hat{x}_0)^2 \tag{5.17}$$

System noise covariance Q: For the system noise covariance, we use the same value as for P_0, but reduced by the previously determined measurement noise, as shown in (5.18), where we assume that $P_0 > R$ always holds. The rationale behind this is that P_0, stemming from the observation during the dead time, should reflect both the system noise (Q) and the measurement noise (R). Note that while Q and R are constant in our approach, P_t is adapted by the EKF over time, so the relationship $Q = P_t - R$ only holds for $t = 0$. Afterwards, during the course of the operation of the EKF, as the EKF converges [143], P decreases over time.

$$Q = P_0 - R \tag{5.18}$$

After the dead time, we define an *ease-in time*, during which the EKF is executed, but its estimates are not yet used. Only after this second phase of the bootstrapping process, the EKF estimates are used for scaling decisions. The time durations used for both the dead time and the ease-in time are parameterizable. In our preliminary experiments, we have found that 10 seconds are sufficient for both parameters.

5.1.5 Approach Analysis

Parameters

We assume that the metrics selected for scaling (contained in the vector z_t) are decided in advance. For instance, in our evaluation in Section 5.2, we use CPU and memory utilization. Furthermore, we assume that the *input* to the system (u_t), is also defined in advance. In our evaluation, D and ΔD constitute this input. The measurements z_t are assumed to be performed with a certain accuracy, defined by a zero-mean Gaussian noise with covariance R. We show in Section 5.1.4 how to determine this parameter, and evaluate this in Section 5.2.3. Furthermore, the input transition function f constitutes a parameter of our approach. In our evaluation, we use the linear function $x + a \cdot D + b \cdot \Delta D$, where a and b are parameters determining the sensitivity of the EKF-based filter to the input data rate. Finally, the state x_t itself is assumed to be subject to zero-mean Gaussian system noise with covariance Q. Like R, Q constitutes a parameter, and in Section 5.1.4, we show how to determine its value. Finally, the duration of both the dead time and the ease-in time, also described in Section 5.1.4, constitute parameters relevant to the bootstrapping process.

Time and Space Complexity

In the following, n denotes the number of elements of the system state vector x, and m denotes the number of elements in the measurement vector z. An EKF iteration (*predict-update*) is required every time new measurements are available. We use a measurement frequency of 2 Hz to remain well below the time required to spin up operator instances, and provide the EKF with sufficiently frequent data. The *predict* step, shown in (5.10)–(5.12), entails the application of f (an $m \times n$ operation), the estimation of z using h (an $n \times n$ operation), and the calculation of P_t (multiple $n \times n$ operations). The *update* step, shown in (5.13)–(5.14), consists of the calculation of the Kalman gain G_t (one $1 \times n$ and multiple $n \times n$ operations), and the estimation of the new system state $\hat{x_t}$, consisting of two $1 \times n$ and one $n \times n$ operation. In summary, the EKF computation time is in $\mathcal{O}(n^2 m)$.

With regard to space, EKFs have the benefit of not keeping history, and therefore the EKF state size is constant over time. It consists of the two state estimation vectors $\hat{x_t}^*$ and $\hat{x_t}$ (cardinality n), the measurement estimation vector $\hat{z}$ (cardinality m), and the matrices P and G_t (cardinality $n \times n$). Overall, the space required for the EKF is in $\mathcal{O}(n^2 + m)$.

5.2 Evaluation

We perform a series of experiments to evaluate our solution. In the following, we describe the testbed, the experimental workload, and our evaluation method.

5.2.1 Experimental Testbed

As outlined in detail in Section 5.2.2, the experiments involve the parallel stream processing of large amounts of images using a private cloud platform. This platform is an OpenStack instance using Kernel-Based Virtual Machines (KVM) running on eight nodes, each with four Intel Xeon E3-1230 v6 CPU cores. In total, 128 GB of memory are available. The DSP operator instances consist of Java applications created for this experiment, using ImageJ[4]. The system of operator instances constitutes our experimental DSP platform.

Operator instances are executed on VMs, where each VM is exclusively assigned one physical core in order to avoid tainting the measured data with adverse artifacts stemming from memory caching, CPU scheduling, and context switches. Depending on the scaling decision, we either spin up additional VMs with new operator instances, or spin down VMs. In this evaluation, we only regard one operator type, with potentially multiple operator instances. We use stateful operator instances, therefore, state must be managed during scaling (see Section 5.2.2). Including both VM boot time and state transfer, preliminary experiments have shown that an operator instance takes between 5 and 25

[4] `https://imagej.nih.gov/ij/`

seconds to be ready. Once available, running operator instances are supplied with images on a round-robin basis from an input queue. While the input queue serves data to the operator instances in First In, First Out (FIFO) order, overall, FIFO order is not guaranteed, since instances might process data at different speeds. We perform no re-synchronization after processing, as our workload processes images individually.

5.2.2 Workload

For our workload, we use images submitted to the DSP for processing, which is a well-known DSP use case, e.g., [271], as well as queries for certain pixel metrics gathered during the stream processing of the images.

DSP Operators

We vary the amount of images submitted to a given operator according to patterns in three distinct workload scenarios. As described in detail in the example scenario for this chapter, the images processed are taken from a real-world biomedical engineering use case [137]. Images of biological cells, obtained from tissue samples and taken using fluorescence microscopy, are analyzed for certain properties. Each image has around 1.7 megapixels, is originally in Tagged Image File Format (TIFF) format, and around 1.3 MB in size.

Furthermore, each image is given Cartesian (X/Y) coordinates determining its position within the overall tissue sample. It is the task of the operators to apply a set of image filters (Gaussian blur, split into RGB channels, edge detection, and object count). In the real-world use case, these filters are used to count biological cells with a given fluorescent marker.

Queries

With an average rate of 1 per 100 images, we also submit queries, where details about pixel intensities regarding a randomly chosen X/Y position in the last 1000 images are requested from the operator. The query consists of the X/Y position, and the return value is a vector containing the last 1000 pixel intensities. This is done to calculate the distribution of pixel intensities for a specific pixel, allowing to assess the significance of this particular X/Y position for the overall result. Therefore, the operator is required to maintain state containing this information, namely a sliding window of length 1000 (count-based, slide of 1).

Furthermore, when scaling out and in, operator instances are made responsible for a given region within the X/Y plane of the overall cell tissue. For this, the plane is split into equal parts, and each operator instance is assigned one of these parts (i.e., sharding by Cartesian coordinates). Therefore, each scaling operation involves migration of state between operator instances. For the sake of evaluation simplicity, we assume no failures (lost state or intermittent disconnection).

SLA Limitations

We impose SLA limitations for both the processing of the images, as well as the queries. The SLA for images is a maximum processing time of 5 seconds, while the SLA limitation for queries is one second. Any processing duration in excess of these deadlines poses a SLA violation.

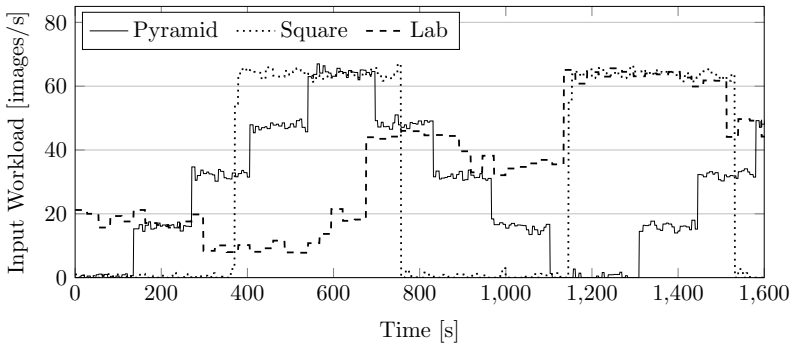

Figure 5.7: Excerpt of the workload scenarios used in the evaluation

The workload is varied according to three different scenarios:

Pyramid: This scenario is generated synthetically using a step-wise increase of the amount of images submitted to the operator input queue, followed by a likewise reduction of this amount, and a repetition of this process for the entire duration of the simulation run. This generates a pyramid-like data rate pattern.

In this work, we use 0 and 60 as the minimum and maximum amount of images per second, respectively. The increase per step is 15 images per second, and each level is held for 130 seconds. These values are chosen to create an easily measureable pattern of usage, while using data rates and timings similar to the real data trace (*Lab*) described below.

Square: Another synthetic pattern is generated by alternating between a low amount of images, and a high amount of images per second used for input to the operator. In this work, we use 1 and 65 as the minimum and maximum amount of images, perform instant changes to the data rate, and hold both values, i.e., 1 and 65, for 370 seconds. This pattern allows us to analyze the impact of very sudden changes in workload on the filters.

Lab: This scenario stems from the used dataset and represents the amount of data processed in the lab. It is not synthesized, but represents real-world observation of the amount of recorded images. This real-life pattern contains both rapid and smooth changes and allows us to measure the behavior of our solution in a usage pattern close to a real-world scenario.

Figure 5.7 shows an excerpt of the input workload pattern for all three scenarios. We use the first two scenarios, i.e., the synthetic patterns *Pyramid* and *Square*, to analyze in detail the performance of our algorithm in extreme cases, such as the rapid increase or decrease of arriving data. We then additionally use the third scenario, i.e., the *Lab* scenario, to verify applicability in a real-world setup.

5.2.3 Evaluation Methodology

We perform the experiment in various configurations, and repeat each configuration 20 times (20 *runs*). An average of the measured values is recorded together with the standard deviation σ. One run lasts 2700 seconds (45 minutes).

Filters

We use three filters in our experiments, consisting of the presented EKF-based approach (denoted as EKF) and two baselines for comparison (denoted as PURE and GW). The PURE filter is the identity function, i.e., no value filtering is used. The GW filter uses the Generalized Weierstrass (GW) transform. The GW transform adds the variance parameter t to the standard Weierstrass transform [272]. This parameter influences the *radius* (variation) of the smoothing effect provided by the Gaussian kernel. The kernel used in the convolution is shown in (5.19), where x represents the time ordinate and t denotes the variance. This notion is in line with existing literature [272].

$$\frac{1}{\sqrt{2\pi t}} e^{-\frac{x^2}{2t}} \tag{5.19}$$

For the implementation, we use a rectangle window function. Due to the low measurement frequency compared to the computational capacity (2 Hz), a sufficiently large window size is feasible and has no significant impact on the result, since the weights for very old measurements are marginally low. We use a window size of 1 minute.

The parameter t is dependent on the system used, and is best set to a value allowing for sufficient smoothing while minimizing the delay of edge detection. In the frequency domain of the Fourier analysis of our data, no signal distinguishable from noise is present above 0.3 Hz. We therefore set $t = 3^2 = 9$, i.e., the wavelength for 0.3 Hz in seconds, squared for variance. A discussion of Gaussian kernels and their usage within signal processing can be found in [176].

Since we perform the filtering live and cannot access future values for our calculation, only the left side of the symmetrical kernel is in effect, denoted by the range $x \leq 0$ and containing past measurements.

A major flaw of all algorithms using linear smoothing methods such as Gaussian-type transforms like the GW transform is the fact that due to the averaging performed in these algorithms, they also smooth out edges in the signal. In elastic stream processing, this means that changes in the data rate are not detected immediately, and thus scaling operations are delayed by design. Furthermore, such algorithms only provide decisions whether to scale out or in, while the EKF also indicates how many additional instances are required. This is due to the inclusion of extrinsic metrics by the EKF, such as the input data rate.

All filters used in the evaluation, i.e., PURE, GW, and EKF, are presented with measurements of the system state. In our scenario, we measure the

Table 5.2: Sensitivity analysis for Θ^- and Θ^+, best result underlined

		Θ^+					
		50%	60%	70%	80%	90%	100%
Θ^-	40%	65,144	63,596	63,554	61,160	69,434	67,850
	50%		63,632	61,484	<u>60,056</u>	62,450	67,244
	60%			61,370	65,000	65,678	69,098
	70%				67,628	69,668	75,818
	80%					79,508	80,168
	90%						79,154

CPU and memory utilization, however, the approach is generally applicable to any (numeric) metrics. The filtered version of the measurements, reflecting the approximated internal system state, is then used to reach scaling decisions. We use threshold-based scaling to evaluate both filters, a technique commonly used in literature [59, 131, 193]. Scaling is performed based on the average value of all operator instances, taking into account every metric. Scaling out is performed if at least one metric is above the scale-out threshold. Scaling in is performed if all metrics are below the scale-in threshold. This is used to avoid bottlenecks stemming from single metrics.

Scaling Thresholds Sensitivity Analysis

We therefore define two thresholds, Θ^-, representing the scale-in threshold, and Θ^+, representing the scale-out threshold. There are various means of choosing thresholds, including automatic learning of parameters. Such learning is not within the scope of this work, but existing techniques, e.g., [189], could be integrated into our approach. Instead, we perform a 20-fold sensitivity analysis to identify both thresholds. The sensitivity analysis is performed for all filters (PURE, EKF, GW) using the three evaluation scenarios (Pyramid, Square, Lab), thus covering the entire parameter domain of this evaluation. Only meaningful values for Θ^- and Θ^+, i.e., $\Theta^- < \Theta^+$ are used. For all threshold combinations, we measure the resulting amount of SLA violations. Table 5.2 shows the resulting average numbers of SLA violations for all filters and scenarios. All standard deviation values σ are below 100 and are not shown. We observe that $\Theta^+ = 80\%$ consistently provides the optimal results. Furthermore, $\Theta^- = 50\%$ provides the optimal results, however, with all three filters, $\Theta^- = 40\%$ is a close runner-up, and we therefore choose $\Theta^- = 45\%$ for our evaluation. Similar values are used in existing literature [116, 154].

CPU Measurement Noise Analysis

In addition, we provide experimental verification for the assumption proposed in Section 5.1.3, according to which the measurement noise of CPU load follows a zero-mean Gaussian distribution. By using FakeLoad [237] in preliminary experiments, we verify that the variance of the measurement is indeed distributed in a sufficiently uniform manner. We perform a 20-fold sensitivity analysis, and while the measured σ does vary, the skew we encounter is not significant

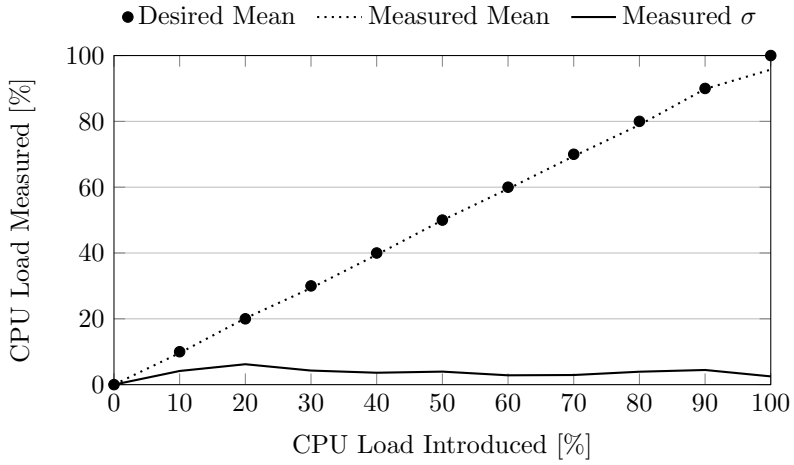

Figure 5.8: CPU load measurement noise analysis

to our evaluation. Figure 5.8 shows the mean data of these measurements. Measurements for memory utilization show almost no measurement noise.

5.3 Experiments and Results

As described in Section 5.2, we evaluate our approach using a total of nine experiment configurations (three filters, three workload scenarios). Each configuration is executed 20 times, and each run lasts 45 minutes. In total, 135 hours of execution time are required for our scenarios.

During the experiments, we record the operators' CPU and memory utilization, the amount of running VMs, the image processing durations, and the SLA violations (see Section 5.2.2). For these metrics, we use the average of all 20 runs for each configuration in order to smooth out measurement noise introduced by the experimentation testbed.

We show an example of individual runs to demonstrate the overall functionality of our evaluation approach in Section 5.3.1. In Section 5.3.2, we present the aggregate results of our experiment runs. We discuss the results in Section 5.3.3.

5.3.1 Exemplary Runs

We first show results from excerpts of individual results, in order to demonstrate the functionality of our EKF approach and its effects on the scaling behavior of the system. Figure 5.9 shows an excerpt of three experiment runs. All three runs use the *Pyramid* workload scenario, and apply either the PURE, GW, or EKF filter. For all three runs, we show the resulting amount of running VMs on the left ordinate. Additionally, for reference, we show the input workload (in

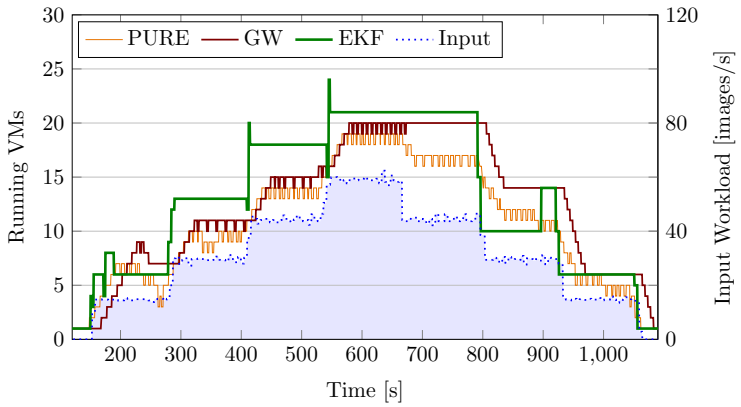

Figure 5.9: Running VMs in the *Pyramid* experiment (left ordinate), and input load (right ordinate)

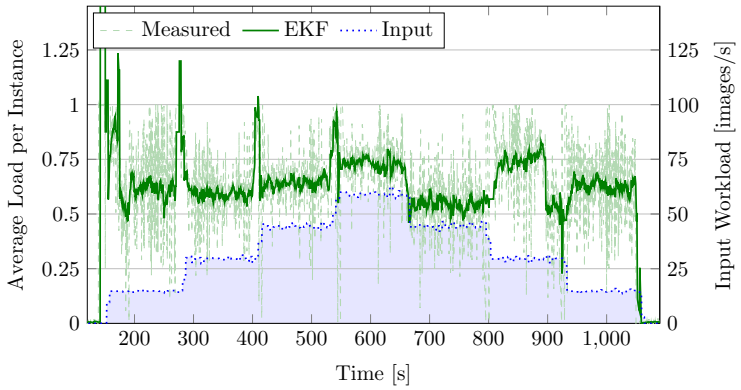

Figure 5.10: Average CPU load in the *Pyramid* experiment (left ordinate), and input load (right ordinate)

images per second) on the right ordinate. The input workload clearly shows the pyramid-like pattern described in Section 5.2.2.

We make the following observations during the analysis of this excerpt:

- GW shows the anticipated delay in reaction (e.g., around $t = 200$, $t = 300$, $t = 410$), which is due to the fact that GW can only provide scale-out or scale-in instructions, but cannot dictate how many VMs are to be spun up or down. Since PURE reacts to changes directly, such delay is not observed for PURE.

- PURE shows excessive scaling operations, reflected in variation of VM count, which is due to the relatively high fluctuation of measurements. GW also suffers from such fluctuations, albeit to a lesser degree.

- GW suffers from overshoot when transitioning from a no-load to a load condition (seen around $t = 220$). This is again due to the fact that GW only gives a scale-out response if Θ^+ is exceeded by any metric. As VMs are spun up, Θ^+ remains exceeded, and especially in the initial start of load ($t = 170$), this causes overshoot.
- EKF shows numerous spikes, where an excessive number of VMs is activated for a load increase. While these spikes are unfavorable, their amount is negligible compared to the frequent scaling of GW and PURE. Further fine-tuning of the state transition function could be used to further reduce these artifacts.
- The crucial advantage of EKF, its potential to not only provide scale-out and scale-in instructions, but also to dictate how many VMs are required to be spun up or down, can be observed. The EKF VM amount, once settled, remains mostly stable.
- EKF, similarly to GW, also exhibits a minor amount of fluctuation when transitioning from a no-load to a load condition. Several scale-out and also some scale-in operations take place around $t = 170$.
- However, EKF settles in a significantly more stable way, and seldom requires correction; an example for this can be seen around $t = 880$. While EKF also shows visible overshoot, this overshoot is minor, and is corrected very soon.
- It is visible that EKF often uses more VMs than GW and PURE, which is caused by its parameters a and b, i.e., the sensitivity to the input data rate. As we see later in Section 5.3.2, while this effect causes more immediate cost due to an increase of VM time required, it leads to substantial reduction of SLA violations and total processing time.
- GW and EKF cause the system to react in an asymmetrical way when considering scale-out and scale-in operations. This is visible when observing the first reduction of workload around $t = 650$. In both cases, the system does not start scaling in until the next workload reduction. This behavior is due to the nature of threshold-based scaling, where a given demand can have multiple different scales while keeping the system load within the thresholds.
- PURE does not exhibit this asymmetry, which we attribute to its frequent scaling operations. PURE is therefore more likely to traverse across its thresholds and does not “linger” in a scaled-out state.

In Figure 5.10, we study in detail the behavior of EKF during the run shown in Figure 5.9. Since during most of the experiments, the CPU load was the factor

limiting scaling operations, we focus on this metric in this analysis. On the left ordinate, we show the measured CPU load, and the EKF-filtered measurement. Both of these numbers are represented averaged over all operator instances. In addition, like in Figure 5.9, for reference, we show the input workload on the right ordinate.

Here, we make the following observations:

- Generally, EKF provides the expected smoothing of measurements.
- While mostly ignoring noise, EKF reacts promptly to changes caused by actual workload increase. In some situations, EKF yields values larger than 1.0, which, in addition to the scale-out decision itself, provides an indication of how many more instances are required.
- We see that the correction around $t = 880$, mentioned in the previous findings, is due the load being close to Θ^+ after the workload decrease at $t = 800$, and finally reaching Θ^+ at $t = 880$, where the aforementioned correction takes place, scale-out is performed, and the average load drops again until $t = 920$.

Summarizing the findings from the exemplary runs, we confirm that the EKF-based filter is working as intended, and the results are qualitatively consistent with the expectations. While naturally the performance of GW could be further increased, especially the low-amplitude fluctuations could be further reduced by fine-tuning parameters, the overall aspects—that is, the delayed response, overshoot, and overall fluctuation of GW—remain.

5.3.2 Aggregate Results

In this section, we present the overall results of our experiments, averaged over 20 runs. We measure the total VM time consumed (measured per second, expressed in hours), the amount of scaling events, the average processing time for images and state queries, and the amount of SLA violations.

Tables 5.3, 5.4, and 5.5 show the results for *Pyramid*, *Square*, and *Lab*, respectively. Since we cannot assume equal variances, we use Welch's t-tests [262] for determining statistical significance. We perform a test for each pair of values measured using PURE, GW, and EKF. We can reject H_0 (i.e., claim significance) for all value pairs except the total VM time using *Square* with GW and EKF (Table 5.4, first row, GW and EKF), where we cannot reject H_0. The p-value is 0.1092, i.e., rejecting H_0 would yield a 10.92% likelihood of a type I error. For all other tests, where we can reject H_0, the p-values are less than 0.004. Therefore, the following observations are based on statistically significant results.

Table 5.3: Aggregate results for the *Pyramid* scenario, lowest results underlined

Metric	PURE (σ)	GW (σ)	EKF (σ)
Total VM Time [1000 h]	8.0 (0.2)	8.4 (0.2)	8.9 (0.3)
Scaling Events	420.6 (11.5)	304.6 (10.5)	58.5 (8.5)
Image Processing Time [s]	2.7 (0.1)	2.1 (0.2)	1.5 (0.7)
Query Processing Time [s]	0.6 (0.1)	0.5 (0.1)	0.3 (0.2)
SLA Violations [1000]	53.8 (0.5)	51.7 (0.7)	47.1 (1.3)

Table 5.4: Aggregate results for the *Square* scenario, lowest results underlined

Metric	PURE (σ)	GW (σ)	EKF (σ)
Total VM Time [1000 h]	8.6 (0.2)	9.0 (0.2)	9.2 (0.5)
Scaling Events	326.1 (3.3)	294.2 (2.4)	28.5 (13.5)
Image Processing Time [s]	4.6 (0.1)	3.9 (0.2)	1.8 (0.1)
Query Processing Time [s]	0.9 (0.1)	0.8 (0.1)	0.3 (0.1)
SLA Violations [1000]	63.2 (0.4)	63.9 (0.2)	47.6 (2.6)

Table 5.5: Aggregate results for the *Lab* scenario, lowest results are underlined

Metric	PURE (σ)	GW (σ)	EKF (σ)
Total VM Time [1000 h]	9.9 (0.2)	10.6 (0.4)	12.1 (1.0)
Scaling Events	423.45 (12.6)	288.0 (18.2)	54.6 (4.2)
Image Processing Time [s]	3.0 (0.1)	2.7 (0.1)	1.6 (0.4)
Query Processing Time [s]	0.8 (0.1)	0.5 (0.1)	0.3 (0.1)
SLA Violations [1000]	69.9 (0.9)	67.0 (0.9)	63.8 (3.6)

Standard Deviation, Mean Values

We note that the standard deviations of all measurements are relatively low (reflected in the fact that all p-values except one are below 0.004), which indicates consistency in our experimental testbed. While the order of standard deviations varies from scenario to scenario (e.g., for the amount of scaling events recorded), due to low standard deviation, this does not pose a risk to our evaluation.

Comparing the measured numbers themselves (i.e., the means), we observe that all scenarios show consistent results. EKF performs best for all metrics except the total VM time consumed. This means that EKF provides a reduction of scaling events, a decreased workload processing time, and less SLA violations, at the cost of VM time. We provide a detailed break-even analysis for this trade-off in Section 5.3.3.

For all metrics, the order between PURE, GW, and EKF is consistent for all scenarios, i.e., GW values are always between PURE and EKF. The only exception is the number of SLA violations in the *Square* scenario, shown below.

Relative Changes

When analyzing percental changes for a metric, we provide the change of EKF compared to GW, followed by the change of EKF compared to PURE in parentheses. For instance, a reduction of scaling events of 80.1% (86.1%) denotes that EKF provides a 80.1% reduction of events compared to GW, and a 86.1% reduction compared to PURE.

In all three scenarios, the total VM time is the highest for EKF, and the lowest for PURE. The effect is strongest in the *Lab* scenario, where the increase is 14.1% (22.2%), followed by *Pyramid*, where the increase is 6.0% (11.3%). The lowest increase, 2.2% (7.0%), is seen for *Square*. This indicates that abrupt, step-wise changes, followed by constant load, as present in *Pyramid* and even more extremely in *Square*, work in favor of EKF, while lower, constant changes in workload increase the additional VM time consumed by EKF. In total, EKF causes an increased VM time of 7.8% (13.9%).

The amount of scaling events is drastically lower for EKF in all scenarios. For *Pyramid*, EKF reduces scaling events by 80.1% (86.1%). For *Square*, the reduction is 90.8% (91.3%), and for *Lab* it is 81.0% (87.1%). Again, *Square* causes EKF to have the strongest effect, however, also the lowest change of 80.1% for *Pyramid* is drastic. In total, EKF causes a decrease in amount of scaling events by 84.0% (87.9%).

The image and query processing times are also lowered by EKF, with a higher impact seen for image processing times. The following figures describe the processing times of all operations, i.e., image processing and state queries. For *Pyramid*, the decrease of processing time is 28.6% (44.5%). For *Square*, the decrease is 53.9% (60.9%). For *Lab*, it is 40.7% (46.7%). Therefore, in the case of processing times, the effect of rapid changes does not seem to have an impact comparable with the previously discussed metrics. In total, operation time is decreased by 43.7% (52.4%).

Finally, we inspect the impact of EKF on the amount of SLA violations. The reduction caused by EKF for *Pyramid* is 8.9% (12.5%), the reduction for *Square* is 25.5% (24.7%), and for *Lab*, it is 4.8% (8.7%). The total reduction of violations is 13.2% (15.2%). Here, again, the highest reduction is seen for *Square*, indicating an impact of rapid changes on the reduction of SLA violations by EKF.

5.3.3 Discussion and Cost Analysis

In the results presented above, we show that using EKF reduces three metrics (scaling events, processing time, and SLA violations), but increases the total VM time consumed. While the reductions provided are substantial, especially with regard to the processing times and the amount of scaling events, the increase of VM time implies a trade-off between increased VM time on the one hand, and a reduction of the remaining metrics on the other hand.

Cloud providers currently do not charge for spin-up and spin-down of machines. Therefore, the scaling events cannot be assumed to cause direct cost but nevertheless lead to higher processing times. In fact, increased processing time usually implies increased cost, either by reducing overall revenue, or by incurring SLA violation penalties.

VM Cost versus SLA Cost

We recall the total increase of 7.8% of VM time, and the reduction of SLA violations by 13.2%, compared to GW. We assume cost of c_v per VM hour, and cost of c_s per SLA violation (i.e., penalty), and formulate the inequation shown in (5.20), stating that the cost caused by additional VM time must be lower than the cost saved by reducing SLA violations.

$$1.078\, c_v < 1.132\, c_s \tag{5.20}$$

From this, we can deduce the inequation shown in (5.21), denoting that the break-even point for EKF is when VM hours are less expensive than SLA violations, with an additional margin of 5%. At the time of performing this evaluation (December 2018), the Google Cloud Platform price for one core hour (Frankfurt) is \$0.0612. Assuming this price, EKF is profitable if the SLA violation penalty is more than \$0.059 per violating image or query.

$$c_v < 1.05\, c_s \tag{5.21}$$

VM Cost versus Operation Revenue

Alternatively, we calculate the break-even point comparing VM hours to operation processing times. We recall the reduction of processing time by 43.7%, compared to GW. This results in an increase of $\frac{1}{1-0.437} = 1.776$, i.e., by 77.6% of processed operations. Again assuming cost of c_v per VM hour, and revenue of r_p per processed operation, we formulate the inequation shown in (5.23), denoting the break-even point for EKF compared to GW. We see that EKF reduces cost if the cost for one VM hour is less than the revenue per operation by a factor of 1.6475. Assuming again \$0.0612 per VM hour, EKF is profitable if the revenue per processed element is higher than \$0.038.

$$1.078\, c_v < 1.776\, r_p \tag{5.22}$$

$$c_v < 1.6475\, r_p \tag{5.23}$$

Based on our experiments, we have shown realistic break-even points for EKF compared to GW, providing a cost-efficient operating range with regard to VM hour cost, SLA violation penalties and operation processing revenue. We use two types of workload in this evaluation. Additional types or complexities of queries could be used, but, depending on their exact form, might require additional metrics to be added to the system state described in Section 5.2.3.

Summarizing our evaluation results, we see that using EKF-based filtering has numerous benefits. Since the EKF includes extrinsic metrics, an estimation of the number of additional VMs is provided, instead of just scale-out or scale-in decisions. This leads to more accurate and less frequent scaling operations, to a considerable reduction of processing time, and to a decrease of SLA violations.

5.4 Related Work

In this section, we discuss the state of the art with respect to our fundamental assumptions in general, regarding existing approaches to scaling in DSP systems, including time series analysis and filtering,

Scaling Overhead As a fundamental assumption for our work, we state that computational overhead caused by scaling induces significant cost. The impact of overhead from scaling of cloud resources has been studied by Corradi et al. [68] (in the context of overhead within cloud data centers) and by Mao et al. [185] (in the context of auto-scaling in cloud workflows). The common result is that indeed, such overhead has significant impact and should be kept to a minimum. In other literature, focus is put on overhead caused not by the scaling itself, but by the decision-making. Computational effort required to solve optimization problems can become quite high [74, 256], especially when using techniques such as MILP [180]. This complexity is somewhat reduced when dealing with ML techniques such as ANNs [140, 204]. As we show in Section 5.1.5, EKF incurs relatively low computational overhead.

Scaling in DSP Next, we consider scaling in DSP systems, a topic thoroughly researched and surveyed in the literature. Work by Mencagli et al. [193] uses the Model-based Predictive Control (MPC) technique to create a trade-off between reconfiguration stability and amplitude. While the context (DSP) is the same, and the aim (reduction of reconfiguration overhead) is similar to ours (reduction of the amount of scaling operations), the authors focus on the use of a distributed and cooperative approach, while we focus on the usage of multiple data sources and reduction of noise.

Floratou et al. [98] propose Dhalion, a self-healing and self-regulating extension for streaming systems, and implement it on top of Twitter Heron. The authors present a framework where metrics are used to detect certain symptoms of declining system health. Dhalion then automatically uses diagnosers to determine possible reasons (diagnoses) for decline in system health, and invokes resolvers to attempt to bring the system back to health. Dhalion features self-monitoring by checking whether actions taken have indeed resolved a symptom, and uses a learning mechanism to blacklist resolutions not contributing to the symptom's resolution. The EKF-based filtering presented in this chapter can be used in combination with Dhalion.

Abadi et al. [2] present the Borealis DSP engine, along with a flexible and QoS-based optimization model. However, while certain intrinsic values are measured (e.g., CPU, disk), the scaling mechanisms presented do not take into account extrinsic metrics such as the rate of input data. No detailed information is given about whether pre-processing of recorded data (e.g., de-noising) is used.

Input Data Rate

The usage of input data rate for scaling decisions has repeatedly been considered in literature [129, 156, 269], as was using threshold-based systems to reach scaling decisions [59, 131]. All of those approaches, however, suffer from the overhead problem described in the introduction to this chapter. Some research has been conducted specifically to tackle this problem of overhead due to volatile input. A general recommendation is the usage of low-pass filters [68]. Another example of linear filters is found in the work by Gong et al. [108], where scaling decisions are based on a Fast Fourier Transform (FFT) and pattern recognition. To avoid overhead cost, the authors use a delayed scaling mechanism, i.e., hysteresis. In contrast to the work at hand, Gong et al. do not include extrinsic (environment) metrics, such as the input data rate. Instead, only hysteresis is used to perform smoothing of metrics. The filtering presented in our work can therefore be used as a stage prior to the scaling mechanism presented by Gong et al.

Time Series Analysis

In our work, we use time series analysis, which is often used for scaling in literature [178]. For instance, this is achieved by creating an auto-scaling algorithm using pattern matching [58], or using wavelet analysis [204]. This is complimented by either using ANNs for proactive and predictive analysis [140, 204], or reinforcement learning [82]. As an interesting approach, a combination of predictive and reactive approaches has been used in [101]. Predictive elements are used for coarse, long-term time scales, while reactive provisioning handles fine-grained, short-term peaks. However, none of these approaches use extrinsic metrics as a data source for reaching scaling decisions.

To the best of our knowledge, apart from our own preliminary work [33], where we outline two approaches extending simple threshold-based scaling, EKF and TVD, only the approach by Gandhi et al. [102] takes into account EKF for resource scaling. In their study, Gandhi et al. propose a performance model using queuing-theoretic principles to predict the trade-off of various scaling decisions. While our approach is situated in the domain of DSP, the authors focus on scaling in cloud computing. The authors employ EKF-based filtering to dynamically infer system parameters required by the performance model. The experimental evaluation is similar to the one presented in Section 5.2. Furthermore, the authors' approach uses EKF to increase robustness, which is also comparable to our approach, albeit in the case of the study by Gandhi et al., the robustness is sought against inaccuracies from the model-driven approach, while in our case, EKF is used to reduce noise stemming from inaccuracies originating in the measurement itself.

5.5 Summary

In this chapter, we have proposed EKF-based filtering for scaling in DSP systems, to reduce the amount of scaling operations and cost caused by SLA violations.

We presented a model in which a time series of measurements of system state is filtered using EKF, and provided details to the application of EKF. We created a system capable of unifying intrinsic measurements (e.g., CPU and memory utilization) with extrinsic measurements (e.g., incoming data rate). The resulting filter quickly reacts to changes in the environment, while minimizing sensitivity to both process and measurement noise. We utilized these filtered values to reach scaling decisions.

We have evaluated our work using a real-world dataset and workload to run experiments, both showing the characteristics of individuals runs, and comparing aggregated numbers to two baselines. While the VM time increased by up to 13.9%, we see an overall reduction of the number of scaling events by up to 87.9%, a reduction of processing time by up to 52.4%, and a decrease in SLA violations by up to 15.2%.

CHAPTER 6

Deterministic Contests in Blockchain Transactions

The approaches presented in the previous chapters all rely on data in the form of measurements for predicting a certain outcome, e.g., by using past system load measurements to predict future system load. However, such approaches are not applicable in situations where data is missing or insufficient. In this chapter, we demonstrate a scenario where the problem of an unpredictable outcome is mitigated by changing the underlying system in a way that makes this outcome deterministic and predictable.

This work is embedded in the context of blockchains, which have recently gained significant interest in both industry and research [158, 280]. Apart from Bitcoin [203], the first blockchain protocol and cryptocurrency, a multitude of different blockchains have emerged, each proposing various features and use cases. In addition to cryptocurrencies native to a specific blockchain (such as Ether, the native currency of the Ethereum blockchain), so-called User-Issued Assets (UIAs) have been created. A popular type of such UIAs are tokens.

The technological variety in the blockchain research space outlines the potential of blockchains. Currently, these blockchains are largely independent and unconnected. Therefore, one of the goals of ongoing research is investigating possible interoperability between blockchains [150]. One means of such interoperability is the design of cross-blockchain token transfer protocols [34].

The main challenge in the development of cross-blockchain technologies is the practical impossibility of verifying the existence of data across blockchains, which we refer to as the Cross-Blockchain Proof Problem (XPP) [35]. This means that novel methods are required for transferring information from one blockchain to another. In this chapter, we present a protocol which allows to trade tokens

independently of a single blockchain. This protocol is called Decentralized Cross-Blockchain Token Transfers (DeXTT) and enables a novel type of tokens, where balances of wallets are stored not only on one blockchain, but on multiple blockchains simultaneously. DeXTT enforces eventual consistency of these token balances across blockchains using incentives, ultimately relaying on observers of a token transfer (so-called *witnesses*) to propagate this transfer information across blockchains.

> **Example Scenario.** Alice is holding digital cryptographic assets in the form of tokens on CHX, a blockchain. She trusts CHX because she believes that it is a trustworthy blockchain and that the technologies used in CHX are solid and hard to attack. In order to purchase goods, Alice wants to transfer a certain amount of these tokens to Bob. However, Bob does not want to accept assets on CHX, because he trusts CHY, another blockchain, more. He believes that the technologies used by CHY are even more secure than those used by CHX. Alice and Bob both do not want to abandon their beliefs about the blockchains' security levels, and they also do not want to involve a centralized third-party exchange service, because this transfer is very sensitive and both parties do not want to risk such a centralized service losing or stealing their funds.
>
> However, thanks to DeXTT, Alice and Bob can trade using a single, cross-blockchain token type. Instead of having to use centralized services to exchange their assets between token types bound to CHX and CHY, respectively, they use a cross-blockchain token which utilizes DeXTT for transfers. This way, both Alice and Bob can exchange assets without having to trust the security level of each other's preferred blockchain.

6.1 Fundamentals

Ecosystem of Blockchains

In the work at hand, we discuss DeXTT, a novel protocol for transferring tokens on a number of blockchain simultaneously. We denote an exemplary instance of a token using this protocol as Cross-Blockchain Token (CBT). Note that multiple instances of such a token can coexist. Our exemplary CBTs are not locked to a single blockchain and can be traded using the DeXTT protocol, which ensures synchronization of token balances across blockchains.

We refer to the set of blockchains participating in this protocol as an *ecosystem of blockchains*. According to our protocol, a wallet $\mathcal{W}_w$ is holding CBTs not only on a given blockchain, but on all blockchains in the ecosystem. Thus, a transfer from $\mathcal{W}_w$ to another wallet $\mathcal{W}_v$ is required to be recorded on all participating blockchains, and there must be consensus among all participating blockchains about the balance of each wallet.

Eventual Consistency

Due to the XPP [35], strict consistency between blockchains is not possible using practical means, since any verification of data between two blockchains would essentially require the nodes of one blockchain to verify blocks of another blockchain. This implies that both the data and the consensus protocol must be shared across blockchains, which is not possible in practice. Therefore, in our proposal, we relax this requirement to eventual consistency, i.e., we accept temporary disagreement with regard to balances, as we show in the following. Eventual consistency is a constraint commonly used in distributed systems [21, 22]. In practice, blockchains themselves only provide eventual consistency, since there is no guarantee when data submitted to the network will be included in a block. Therefore, using eventual consistency for synchronizing data between blockchains is a feasible approach.

For the purpose of this work, we follow the assumption that each party is generally interested in all the blockchains in an ecosystem, and specifically, in the consistency of their balance across all blockchains. This means that all interested parties (i.e., wallet holders) are monitoring all blockchains in the ecosystem, and if a party participates in the protocol on one blockchain, it also participates on all other blockchains. We support this assumption by defining later that any inconsistency in wallet balances between blockchains effectively renders the wallet useless.

DeXTT assumes that non-zero token balances already exist on the involved blockchains. We explicitly do not define the economic aspect of CBTs, i.e., the lifecycle of tokens. Several minting strategies exist, and we provide an overview of such approaches (constant supply, minting rate, etc.) in previous work [34]. Any of these approaches is usable together with DeXTT, since the protocol assumes that tokens already exist.

6.1.1 Cross-Blockchain Balance Consistency

As outlined before, we require eventual consistency between blockchains participating in the proposed protocol. Since due to the XPP, we cannot directly propagate information across blockchains, we require an alternative way to reach consistency across blockchains.

Claim-First Transactions [35]

For this, we propose to achieve eventual consistency using *claim-first transactions* [35]. While traditionally, blockchain transfers disallow claiming tokens before they have been marked as spent, we explicitly decouple this temporal order and allow its reversal, i.e., claiming tokens before spending them. In our case, for a certain period of time, tokens are allowed to exist in the balance of both the sender and the receiver (on different blockchains), namely until the information is propagated to all blockchains. In the presented protocol, we provide a mechanism to enforce eventual spending of the tokens in the sender balance, as described in Section 6.2.

Witnesses

In order to ensure such eventual consistency, we rely on parties observing a transfer to propagate this information across blockchains. These parties are denominated as *witnesses*. A monetary incentive is provided for any witness in order to ensure propagation. We use part of the transferred CBTs for these witness rewards. The main challenge of this approach is the decision which witness receives the reward. Using a first-come-first-serve basis is not feasible, since it is possible that on one blockchain, one witness is the first to propagate the transfer and claim the reward, while on another blockchain, another witness takes this place. This would lead to two different witnesses receiving a reward on two different blockchains, and therefore, to potentially inconsistent balances.

Contestants

In this work, we address this problem by using *deterministic witnesses* [36]. Instead of using a first-come-first-serve reward distribution, we define a *witness contest* which provides incentive for users to participate in the protocol. Its duration is fixed to a validity period, *contestants* (i.e., reward candidates) can register for the contest, and the decision of who wins the contest is made deterministically and predictably by each blockchain at the end of the contest. In Section 6.2, we propose an approach to deciding the winning witness in a way that is fair (i.e., all contestants have the same chance of winning), while at the same time, it is designed to be purely deterministic by definition, and—given the assumptions discussed above—assures all blockchains reach the same decision about assigning witness rewards.

Note that the requirement for participating in the witness contest is the propagation of information about the token transfer. Therefore, the contest is used as a vehicle of cross-blockchain information transfer, with the witness reward used to ensure its reliability.

Our approach therefore solves the problem of assigning witness rewards, which is required as an incentive for observers of a cross-blockchain transfer to propagate this transfer information, ensuring eventual consistency across the ecosystem of blockchains. In contrast to centralized solutions, our contribution provides an entirely decentralized and trustless approach to cross-blockchain transfers.

6.1.2 Cryptographic Signatures and Hashes

In our approach, we make extensive use of cryptographic signatures and hashes, which are essential for blockchains themselves. For instance, the Elliptic Curve Digital Signature Algorithm (ECDSA) algorithm [149] is used by Ethereum for creating and verifying signatures, and is also implemented natively and available to the Ethereum Virtual Machine (EVM) [128]. We use Solidity, the smart contract language of Ethereum, for the reference implementation of DeXTT. However, we note that DeXTT is not limited to Solidity or the EVM, and other blockchains offering signatures and hash algorithms can very well be used. The only crucial property required by our approach is a distribution of hash

values which is approximately uniform. KECCAK256, the hash algorithm used by Ethereum, satisfies this requirement [103], as does the SHA-256 algorithm used by Bitcoin [104].

6.1.3 Notations and Conventions

In the following, we use particular notations for the concise description of certain objects: We denote blockchains as $\mathcal{C}$ with a subscript letter, e.g., $\mathcal{C}_a$. Additionally, we denote wallets as $\mathcal{W}$ with a subscript letter, e.g., $\mathcal{W}_s$, $\mathcal{W}_d$, or $\mathcal{W}_w$. A wallet consists of a pair of corresponding keys, out of which one is a public key, and one is a private key. When referring to a token transfer in general, $\mathcal{W}_s$ is used to denote the source (sending) wallet, $\mathcal{W}_d$ is used to denote the destination (receiving) wallet, and $\mathcal{W}_w$ denotes a witness as discussed in Section 6.1.1. As discussed above—and later demonstrated in Table 6.1—the balance of a wallet is stored across all blockchains.

Transactions

In this work, we use the concept of *transactions* to denote actions executed on a blockchain which modify the blockchain state. We use the expression "$\mathcal{W}_w$ posts the transaction TRANS on $\mathcal{C}_c$" to describe the conceptual protocol. In a scenario where smart contracts are used, this translates to the key pair of $\mathcal{W}_w$ being used to sign a call to the smart contract on blockchain $\mathcal{C}_c$, where the function `trans()` is invoked. For certain transactions, we define preconditions (e.g., sufficient balances), which can be implemented as checks within the smart contract function. The transactions posted by wallets can either originate from the action of a user, or be initiated by a program (e.g., a wallet application) acting autonomously.

To denote our transactions, we use the notation as shown in (6.1), where TRANS is the transaction type used (one out of CLAIM, CONTEST, FINALIZE, VETO, and FINALIZE-VETO), $\mathcal{W}_w$ is the wallet (i.e., the pair of keys) used to sign and post the transaction, a, b, and c denote data contained in the transaction (i.e., the arguments), and σ is the signature when using the private key of $\mathcal{W}_w$ to sign the data $[a, b, c]$. For brevity, we use only σ to denote a multivariate value, e.g., a three-variate ECDSA signature.

$$\mathcal{W}_w : \text{TRANS} \left[a,\, b,\, c \right]_\sigma \tag{6.1}$$

We denote a transfer of x CBTs from $\mathcal{W}_s$ to $\mathcal{W}_d$ as $\mathcal{W}_s \xrightarrow{x} \mathcal{W}_d$. Furthermore, we denote the CBT balance of $\mathcal{W}_w$ recorded on $\mathcal{C}_c$ as $\mathcal{C}_c : \mathcal{W}_w$.

6.2 Decentralized Cross-Blockchain Transfers

In the following, we present the DeXTT protocol, together with an example transaction. In our example, we consider three blockchains participating in

Table 6.1: Initial state of the involved blockchains at $t = 0$

Blockchain $\mathcal{C}_a$	
$\mathcal{W}_s$ balance:	80
$\mathcal{W}_d$ balance:	0
$\mathcal{W}_w$ balance:	0

Blockchain $\mathcal{C}_b$	
$\mathcal{W}_s$ balance:	80
$\mathcal{W}_d$ balance:	0
$\mathcal{W}_w$ balance:	0

Blockchain $\mathcal{C}_c$	
$\mathcal{W}_s$ balance:	80
$\mathcal{W}_d$ balance:	0
$\mathcal{W}_w$ balance:	0

cross-blockchain transfers, $\mathcal{C}_a$, $\mathcal{C}_b$, and $\mathcal{C}_c$. Note, however, that our approach is applicable to an arbitrary number of blockchains. Furthermore, we consider the wallets $\mathcal{W}_s$, $\mathcal{W}_d$, $\mathcal{W}_u$, $\mathcal{W}_v$, and $\mathcal{W}_w$. We assume that initially, $\mathcal{W}_s$ has 80 CBTs, and all other wallets have a balance of zero (see Table 6.1). We furthermore use a fixed reward of 1 CBT for the witness propagating this transaction across the blockchain ecosystem. Note that pro rata fees (e.g., 1% of the transferred CBTs, or an amount selected by the sender) are also possible and the exact fee model is an economic choice. We will discuss this in more detail in Section 6.3.2.

As discussed in Section 6.1.1, claim-first transactions require all blockchains within the ecosystem to maintain and synchronize token balances. Therefore, the initial situation is as depicted in Table 6.1. Balances for $\mathcal{W}_u$ and $\mathcal{W}_v$ are not shown, as they will remain zero throughout the example.

6.2.1 Transfer Initiation

In the following, we assume that $\mathcal{W}_s$ intends to transfer 20 CBTs to $\mathcal{W}_d$, i.e., to reduce the CBT balance of $\mathcal{W}_s$ by 20, increase the CBT balance of $\mathcal{W}_d$ by 19 (20 reduced by 1, the witness reward), and increase the CBT balance of a (yet to be decided) witness wallet by 1. As stated in Section 6.1.1, we only require eventual consistency for this transfer, i.e., a temporary overlap is allowed where $\mathcal{W}_d$ has already received 19 CBT on one blockchain, but the balance of $\mathcal{W}_s$ is still unchanged on another blockchain.

Therefore, $\mathcal{W}_s$ signs this intent, confirming that indeed, 20 CBTs—minus 1 CBT of witness reward—are to be transferred to $\mathcal{W}_d$. Furthermore, we define a validity period for the transfer, which denotes the time during which the witness selection for the transfer has to take place. In our example scenario, this time span lasts for 1 minute, however, this time can be set significantly shorter or longer, depending on the use case. We provide an analysis of the impact of this parameter in Section 6.3.1.

We denote the entirety of the sender's intent using the notation shown in (6.2), where $[t_0, t_1]$ is the validity period, and $[\,\cdot\,]_\alpha$ denotes the signature of the entire content of the brackets by $\mathcal{W}_s$. The resulting signature itself is denoted as α. We use the ECDSA algorithm, natively supported by the EVM, for all signatures.

However, other algorithms can also be used, assuming that their verification is supported on all involved blockchains.

$$\left[\mathcal{W}_s \xrightarrow{x} \mathcal{W}_d,\, t_0,\, t_1 \right]_\alpha \tag{6.2}$$

The data contained in (6.2) is transferred to the receiving wallet $\mathcal{W}_d$. This transfer can happen on any blockchain within the ecosystem, or using an off-chain channel. Since all of the data contained in (6.2) will be published throughout the DeXTT transaction, this channel does not need to be secure, and we do not specifically define any communication means. The receiving wallet then counter-signs the data from (6.2) using its respective private key, yielding the entire Proof of Intent (PoI), as shown in (6.3). Similar to α in (6.2), β in (6.3) denotes the signature of all data contained in the brackets, as signed by $\mathcal{W}_d$.

$$\left[\mathcal{W}_s \xrightarrow{x} \mathcal{W}_d,\, t_0,\, t_1,\, \alpha \right]_\beta \tag{6.3}$$

The PoI contains all information necessary to prove to any blockchain (i.e., to its smart contracts and miners) that the transfer is authorized by the sender and accepted by the receiver. The receiver can now post this PoI using a transaction we call CLAIM. This transaction allows the receiver to publish the PoI in order to later claim the transferred CBTs. The receiver can post this on any blockchain within the ecosystem, and does not need to post it on more than one blockchain, as we will see later. The CLAIM transaction is defined as shown in (6.4).

$$\mathcal{W}_d : \textsc{claim} \left[\mathcal{W}_s \xrightarrow{x} \mathcal{W}_d,\, t_0,\, t_1,\, \alpha \right]_\beta \tag{6.4}$$

The preconditions for the CLAIM transaction are (i) that the PoI is valid (i.e., that the signatures α and β are correct), (ii) that the balance of the source wallet $\mathcal{W}_s$ is sufficient (larger than x), (iii) that the PoI is within its validity period ($t_0 < t < t_1$), and (iv) that no PoI is known to the blockchain on which it is posted with an overlapping validity period and the same source wallet $\mathcal{W}_s$. In other words, a wallet must not sign an outgoing PoI while another outgoing PoI is still pending. This is done in order to prevent a double spending attack, where two PoIs are signed which are conflicting, i.e., which, if both were executed, would reduce the sender's balance below zero.

The purpose of the CLAIM transaction is the publishing of the PoI, which can then be propagated across the blockchain ecosystem as described later.

In our example, we assume that the receiver $\mathcal{W}_d$ posts the CLAIM transaction (containing the PoI) on $\mathcal{C}_a$ as shown in (6.5), where 1 and 61 mark the

Table 6.2: State after PoI publication at $t = 1$

Blockchain $\mathcal{C}_a$		Blockchain $\mathcal{C}_b$		Blockchain $\mathcal{C}_c$	
$\mathcal{W}_s$ balance:	80	$\mathcal{W}_s$ balance:	80	$\mathcal{W}_s$ balance:	80
$\mathcal{W}_d$ balance:	0	$\mathcal{W}_d$ balance:	0	$\mathcal{W}_d$ balance:	0
$\mathcal{W}_w$ balance:	0	$\mathcal{W}_w$ balance:	0	$\mathcal{W}_w$ balance:	0
PoI `0xAA`:					
$\mathcal{W}_s \xrightarrow{20} \mathcal{W}_d$					
$t_1 = 61$					

validity period in seconds (i.e., one minute total validity), `0xAA` is assumed to be the signature α, and `0xBB` is assumed to be the signature β. For brevity, one-byte signatures are used for demonstration in this example. Naturally, in reality, the signature hashes are longer (e.g., 32 bytes for Keccak256).

$$\mathcal{W}_d : \textsc{claim} \left[\mathcal{W}_s \xrightarrow{20} \mathcal{W}_d,\ 1,\ 61,\ \texttt{0xAA} \right]_{\texttt{0xBB}} \tag{6.5}$$

The claim transaction on $\mathcal{C}_a$ changes the blockchain state as shown in Table 6.2. We see that the PoI has been stored within $\mathcal{C}_a$, and is referred to by its signature α. The balances remain unchanged on $\mathcal{C}_a$ because the validity period is not yet concluded, i.e., t_1 is not yet reached. Naturally, since no information has been posted yet to $\mathcal{C}_b$ and $\mathcal{C}_c$, these blockchains also remain unchanged.

6.2.2 Witness Contest

At this point, the information about the intended transfer (the PoI) is only recorded on $\mathcal{C}_a$. However, this information must be propagated to all other blockchains as well to ensure consistency of balances across blockchains. As described in Section 6.1.1, we use a monetary reward provided to observers to ensure this consistency. The observer receiving this witness reward is selected during the witness contest as described in this section.

Any party observing the claim transaction on $\mathcal{C}_a$ can become a contestant, i.e., a candidate for receiving a reward. In order to become a contestant, the party must propagate the PoI across all blockchains in the ecosystem. We define the transaction used for this as contest. This transaction is defined for any arbitrary wallet $\mathcal{W}_o$ as shown in (6.6), where the new signature ω is the result of the contestant $\mathcal{W}_o$ signing the PoI. This signature will later play a role in determining the winner of the witness contest, as described in Section 6.2.3.

$$\mathcal{W}_o : \textsc{contest} \left[\mathcal{W}_s \xrightarrow{x} \mathcal{W}_d,\ t_0,\ t_1,\ \alpha,\ \beta \right]_{\omega} \tag{6.6}$$

The CONTEST transaction can be posted multiple times by various contestants during the validity period. The preconditions are the same as for the CLAIM transaction, i.e., the PoI must be valid and must not violate any other PoI validity period. The only effect of the CLAIM transaction is that the PoI itself and the contestant's participation in the witness contest are recorded on the respective blockchain.

In our example, we assume that $\mathcal{W}_u$ is the first to post a CONTEST transaction on $\mathcal{C}_b$ as shown in (6.7), where again, 1 and 61 denote the validity period, `0xAA` and `0xBB` are the PoI signatures, and `0xC2` is the signature resulting from $\mathcal{W}_u$ signing the PoI. The signature values in this example are chosen arbitrarily in order to demonstrate the subsequent witness contest. Again, one-byte signatures are used for brevity.

$$\mathcal{W}_u : \text{CONTEST} \left[\mathcal{W}_s \xrightarrow{20} \mathcal{W}_d,\ 1,\ 61,\ \texttt{0xAA},\ \texttt{0xBB} \right]_{\texttt{0xC2}} \tag{6.7}$$

Next, we assume that the other observers $\mathcal{W}_v$ and $\mathcal{W}_w$ become contestants by posting similar CONTEST transactions. We assume that the resulting signature ω for $\mathcal{W}_v$ is `0xC3`, and that the signature for $\mathcal{W}_w$ is `0xC1`.

$$\mathcal{W}_v : \text{CONTEST} \left[\mathcal{W}_s \xrightarrow{20} \mathcal{W}_d,\ 1,\ 61,\ \texttt{0xAA},\ \texttt{0xBB} \right]_{\texttt{0xC3}} \tag{6.8}$$

$$\mathcal{W}_w : \text{CONTEST} \left[\mathcal{W}_s \xrightarrow{20} \mathcal{W}_d,\ 1,\ 61,\ \texttt{0xAA},\ \texttt{0xBB} \right]_{\texttt{0xC1}} \tag{6.9}$$

Transactions (6.7–6.9) are eventually posted to $\mathcal{C}_a$, $\mathcal{C}_b$, and $\mathcal{C}_c$. This is because according to the assumption in Section 6.1, every contestant participating in the contest is interested in participating in all blockchains in the ecosystem to maintain consistency of their own wallets.

The state resulting from the three contestants posting to $\mathcal{C}_a$, $\mathcal{C}_b$, and $\mathcal{C}_c$ is shown in Table 6.3. The blockchain maintains a list of contestants together with their ω signature values.

6.2.3 Deterministic Witness Selection

After the expiration of t_1, the witness contest ends, and a winning witness must be selected to be awarded the witness reward. This is performed by the FINALIZE transaction as shown in (6.10), which must be triggered after t_1.

$$\text{FINALIZE} \left[\alpha \right] \tag{6.10}$$

Table 6.3: State during witness contest at $t = 2$

Blockchain $\mathcal{C}_a$	
$\mathcal{W}_s$ balance:	80
$\mathcal{W}_d$ balance:	0
$\mathcal{W}_w$ balance:	0
PoI `0xAA`:	
$\mathcal{W}_s \xrightarrow{20} \mathcal{W}_d$	
$t_1 = 61$	
Contestants:	
$\mathcal{W}_u$ (`0xC2`)	
$\mathcal{W}_v$ (`0xC3`)	
$\mathcal{W}_w$ (`0xC1`)	

Blockchain $\mathcal{C}_b$	
$\mathcal{W}_s$ balance:	80
$\mathcal{W}_d$ balance:	0
$\mathcal{W}_w$ balance:	0
PoI `0xAA`:	
$\mathcal{W}_s \xrightarrow{20} \mathcal{W}_d$	
$t_1 = 61$	
Contestants:	
$\mathcal{W}_u$ (`0xC2`)	
$\mathcal{W}_v$ (`0xC3`)	
$\mathcal{W}_w$ (`0xC1`)	

Blockchain $\mathcal{C}_c$	
$\mathcal{W}_s$ balance:	80
$\mathcal{W}_d$ balance:	0
$\mathcal{W}_w$ balance:	0
PoI `0xAA`:	
$\mathcal{W}_s \xrightarrow{20} \mathcal{W}_d$	
$t_1 = 61$	
Contestants:	
$\mathcal{W}_u$ (`0xC2`)	
$\mathcal{W}_v$ (`0xC3`)	
$\mathcal{W}_w$ (`0xC1`)	

Conceptually, the FINALIZE transaction is purely time-based. It can be triggered by the receiver, by any other party, or using a decentralized solution like the *Ethereum Alarm Clock* [19]. The latter approach has the advantage of being independent of any party's activity. However, for simplicity, in our current approach and the discussion below, we assume that the destination wallet $\mathcal{W}_d$ posts the FINALIZE transaction on each blockchain.

The FINALIZE transaction only requires the parameter α, identifying the PoI, because the blockchain already contains all necessary information about the PoI. The precondition of t_1 being expired ($t > t_1$) is necessary for the FINALIZE transaction to avoid premature finalization.

The effect of the FINALIZE transaction is that the contest for the PoI—referred to by its signature α—is concluded. This means that the winning witness is awarded the witness reward, which, according to Section 6.2, is 1 CBT in our current approach. Furthermore, the conclusion of the contest performs the actual transfer of CBTs, i.e., x CBTs are deducted from the balance of $\mathcal{W}_s$, and $\mathcal{W}_d$ receives $(x - 1)$ CBTs (x reduced by the witness reward). This action is executed on all blockchains, since FINALIZE is posted on all blockchains.

We define the winning witness to be the contestant with the lowest signature ω (i.e., with its value closest to zero). Since this signature cannot be influenced by the contestants (because it is only formed from the PoI data and the contestants' private key), they have no way of increasing their chances of winning a particular contest, except for creating a large number of wallets (private keys).

Such "mining for wallets" is not a violation of our protocol and no threat to its fairness, since doing so is computationally expensive, and therefore creates cost on its own. There exists a break-even point of the witness reward and the cost created by the creation of a large number of wallets [36]. Effectively, this

Table 6.4: Final state after witness contest at $t > 61$

Blockchain $\mathcal{C}_a$	
$\mathcal{W}_s$ balance:	60
$\mathcal{W}_d$ balance:	19
$\mathcal{W}_w$ balance:	1

Blockchain $\mathcal{C}_b$	
$\mathcal{W}_s$ balance:	60
$\mathcal{W}_d$ balance:	19
$\mathcal{W}_w$ balance:	1

Blockchain $\mathcal{C}_c$	
$\mathcal{W}_s$ balance:	60
$\mathcal{W}_d$ balance:	19
$\mathcal{W}_w$ balance:	1

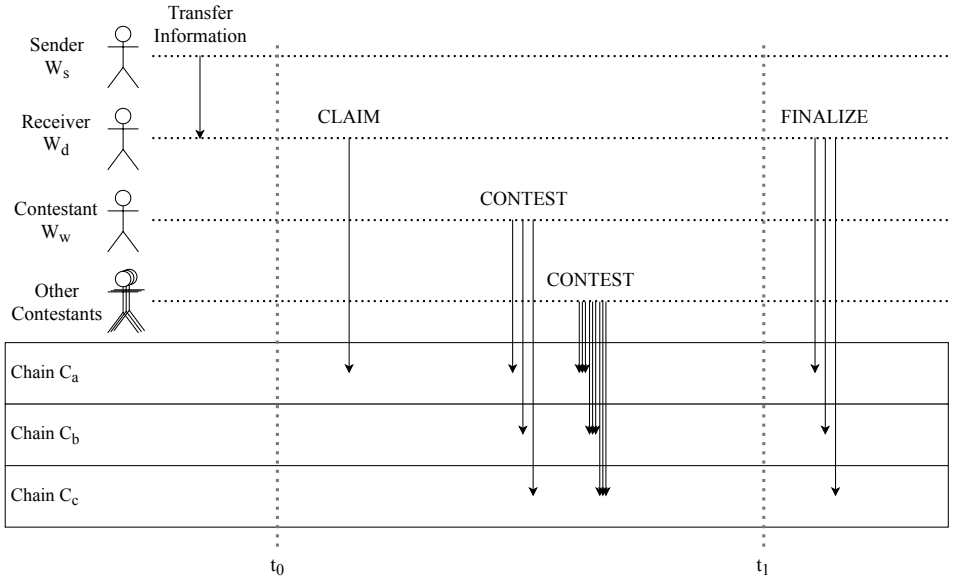

Figure 6.1: Sequence of transactions within a DeXTT transfer

challenge is comparable to mining in Proof of Work (PoW) in that resources, i.e., computing power, can be traded for rewards.

In our example above, the witness with the lowest ω is $\mathcal{W}_w$, with $\omega = \texttt{0xC1}$. Therefore, this witness is awarded the witness reward. The final blockchain state is shown in Table 6.4. The balances of the competing contestants $\mathcal{W}_u$ and $\mathcal{W}_v$—not shown in Table 6.4—remain zero. The expired PoIs are no longer shown for brevity.

Figure 6.1 shows an overview of the transactions posted by various wallets on various blockchains. The contestant which ultimately becomes the winning witness ($\mathcal{W}_w$) is shown separately from all other contestants, since this wallet is later assigned the witness reward. We see that first, the sender $\mathcal{W}_s$ provides the receiver $\mathcal{W}_d$ with the transfer information shown in (6.2). This may happen before or after t_0. Then, not sooner than t_0, the receiver posts a CLAIM transaction to one of the blockchains (in this case, $\mathcal{C}_a$). This is observed by all contestants, which then post CONTEST transactions to all blockchains. Note

that the CONTEST transactions do not have to follow any particular order, and are expected to be posted concurrently by all contestants.

After t_1 expires, the receiver posts the FINALIZE transaction, which finalizes the transfer and deterministically assigns the witness reward to the contest winner (in this case, $\mathcal{W}_w$).

6.2.4 Prevention of Double Spending

A malicious sender might sign two different PoIs conflicting with each other. For instance, a sender owning 10 CBTs might create two PoIs, transferring 8 CBTs each, to two different wallets. Executing these transfers would reduce the sender's balance by 16 CBTs in total, resulting in −6 CBTs.

In order to prevent such behavior, we introduce the VETO transaction. The VETO transaction can be called by any party noticing two conflicting PoIs (i.e., two PoIs with the same source, different destinations, and overlapping validity periods). Since such PoIs are forbidden by definition, the VETO transaction is used to penalize the sender, and to protect the receiver from losing CBTs due to inconsistent balances.

Since the VETO transaction requires incentive, we propose to use the same technique as presented above, i.e., a deterministic contest. Any observer of a PoI conflict can report this conflict using the VETO transaction, and after the expiration of the veto validity period, the observer with the lowest ω signature is assigned a reward.

We therefore define the VETO transaction as shown in (6.11), where α refers to the original PoI, which is known to the blockchain because it has already been posted on a given blockchain, and the remaining data $\mathcal{W}_s$, $\mathcal{W}_{d'}$, x', t'_0, t'_1, and α' describe the new, conflicting PoI.

$$\mathcal{W}_w : \text{VETO} \left[\alpha, \mathcal{W}_s \xrightarrow{x'} \mathcal{W}_{d'}, t'_0, t'_1, \alpha' \right]_\omega \tag{6.11}$$

The VETO transaction, similar to CONTEST, is posted on all participating blockchains. Note that multiple observers can be expected to concurrently post VETO transactions. Therefore, it is possible that on one blockchain, a given PoI (e.g., where $\alpha = \texttt{0x10}$) is posted first, and a second PoI (e.g., where $\alpha' = \texttt{0x20}$) is presented as "conflicting" by a VETO transaction, while on another blockchain, the PoI where $\alpha = \texttt{0x20}$ is posted first, and the PoI with $\alpha' = \texttt{0x10}$ is posted in the VETO transaction as "conflicting". However, in the following, we define a behavior for the VETO transaction that still maintains consistency, regardless of the order of PoIs.

The preconditions for VETO are that α refers to a PoI already known to the blockchain, that the conflicting PoI is valid, and that the two contained PoIs are actually conflicting.

The effects of VETO are as follows: (i) The sender of the conflicting PoIs loses all CBTs, i.e., the balance is set to zero to penalize such protocol-violating behavior. (ii) Any of the two PoIs with a non-expired validity period (i.e., any PoI where $t < t_1$) is canceled. This means that no FINALIZE transaction will be permitted for this PoI, the transfer itself will therefore not be executed, and no witness reward will be assigned. Finally, (iii) a new contest is started, called the *veto contest*. The veto contest is similar to a regular witness contest in that its purpose is the propagation of information (in this case, the information of conflicting PoIs).

In the following, we propose a possible implementation of such veto contest, however, its details (i.e., the definition of its validity period or the reward) are specifics which may be implemented differently.

We propose to use the same reward for the veto contest as for the regular witness contest (in our case, 1 CBT). Since all CBTs held by the sender are destroyed, and only 1 CBT is assigned to the winner of the veto contest, all remaining CBTs are lost. Furthermore, we propose the validity period expiration of the veto contest, t_{VETO}, to be defined as shown in (6.12).

$$t_{\text{VETO}} = \max(t_1, t_1') + \max(t_1 - t_0, t_1' - t_0') \tag{6.12}$$

The definition shown in (6.12) states that the veto contest is valid until a point in time which is found by taking the later expiration time of the conflicting PoIs ($\max(t_1, t_1')$) and adding the longer validity period ($\max(t_1 - t_0, t_1' - t_0')$). This is done to ensure that sufficient time is available for the veto contest. Again, we note that this is an implementation detail and other approaches (e.g., a fixed period) are also possible.

The veto contest is concluded by a FINALIZE-VETO transaction, defined as shown in (6.13).

$$\text{FINALIZE-VETO} \left[\alpha, \alpha' \right] \tag{6.13}$$

The effect of the FINALIZE-VETO transaction is similar to that of the FINALIZE transaction, except that no actual transfer is executed. The witness reward is again assigned to the veto contestant—that is, a wallet posting a VETO transaction—with the lowest ω signature in the VETO transaction. Similar to the FINALIZE transaction, the FINALIZE-VETO transaction can be called by anyone, and in particular, the winning veto contestant has monetary incentive in doing so.

6.3 Evaluation

The approach presented in Section 6.2 introduces transactions which change the state of different blockchains within a blockchain ecosystem, according to given rules. This can be implemented using smart contracts, e.g., using the Solidity language [71]—more specifically, the EVM—on the Ethereum blockchain. We use Solidity to create a reference implementation of the proposed protocol for evaluation[1]. However, other ways of implementing such transactions exist. For instance, instead of using smart contracts (e.g., when dealing with blockchains without such capabilities), one might add backwards-compatible layers on top of blockchains, providing the required capabilities for the transactions presented in this work. A similar approach is used by OmniLayer [266] or CounterParty [69, 70], which add such layers for enhanced features. For this work, however, we use our reference Solidity implementation of DeXTT for evaluation and cost analysis, postponing the integration of approaches such as OmniLayer or CounterParty to future work. Nevertheless, our current evaluation is sufficient to demonstrate the overall functionality of the DeXTT protocol using Solidity smart contracts as well as its conceptual applicability.

In order to evaluate our approach, we investigate its functionality, performance, and cost impact in an ecosystem of blockchains with agents performing repeated token transfers. We achieve these goals by using our reference implementation consisting of Solidity smart contracts, deploying these smart contracts on a number of private Ethereum-based blockchains, and using testing client software to perform transfers with a given rate.

We ensure a reproducible and uniform ecosystem of blockchains by using three `geth` nodes in Proof of Authority (PoA) mode, creating three private blockchains. We choose PoA to achieve an energy-efficient testing and evaluation platform while being able to perform repeated experiments. Note that the consensus algorithm, i.e., PoW, PoA, or Proof of Stake (PoS), defines the behavior of blockchain nodes between each other and maintains data consistency in the network of a given blockchain [279]. However, the smart contract layer is independent of the consensus algorithm. Therefore, our evaluation on PoA is directly applicable to blockchains with any consensus algorithm, including PoW.

The `geth` nodes used in our experiments can be configured, for instance, with regard to block time and Gas limit. We observe the behavior of the live Ethereum blockchain (as of January 2019) and configure our nodes to follow this behavior. Therefore, our nodes are configured to use a block time of 13 s on average, and a Gas limit of 8 million Ethereum Gas, mimicking the live Ethereum chain. We use private chains instead of the Ethereum main chain to enable a high number and low cost of repeatable experiments in an automated fashion without depending on external components, such as Ethereum nodes.

[1] `https://github.com/pantos-io/dextt-prototype`

We use 10 clients constantly and simultaneously initiating transfers within the blockchain ecosystem. This number is chosen as a balance between feasible and reproducible experiments and expected real-world conditions. While it is small compared to evaluations of other classes of distributed systems, we note that the lack of scalability of blockchain technologies is a crucial issue in general, and is seen as one of the main challenges for existing blockchain technologies [126]. We refer to existing literature for a study on how scalability of blockchains can be improved [84, 258].

In our experimental ecosystem, each client constantly transfers random amounts of CBTs to random wallets. If a client owns too few CBTs for a transaction, no transaction is performed until CBTs are available again. After a successful transfer, the client waits for a random time between 15 s and 30 s. Afterwards, the process is repeated indefinitely throughout the entire experiment duration.

We perform two experiment series, as described in the following sections. The first series is used to evaluate DeXTT scalability and the impact of the transfer validity period, and consists of a series of 30-minute experiments, where each individual experiment uses an increased validity period. The second series consists of 20 experiments, again with a duration of 30 min each, used to measure the average cost of a DeXTT transfer.

6.3.1 Scalability and Timing

The DeXTT protocol requires one CLAIM transaction per transfer, and for each transfer, one FINALIZE transaction per blockchain. In addition, each contestant posts one CONTEST transaction to each blockchain. We assume that candidates which no longer have a chance to win the witness contest (because a candidate with a lower signature ω for the given transaction has already posted a CONTEST transaction) do not post CONTEST transactions to avoid cost. Assuming uniform distribution of ω values, as defined in Section 6.1.2, on the average case, each CONTEST transaction halves the space of remaining possible winning signatures ω (because the expected value of the uniform distribution is the arithmetic mean of the domain). Therefore, with each CONTEST transaction, the likelihood of another candidate existing with a lower ω is halved. Following from this, on average, $\log_2 n$ candidates will post a CONTEST transaction, where n is the number of total observers.

Validity Period

Transfer time in the DeXTT protocol is directly impacted by the transfer validity period $[t_0, t_1]$ chosen by the sender. We therefore first evaluate the impact of the validity period. Using too short validity periods leads to corrupted transfers, i.e., transfers which cause permanently inconsistent balances, since observers cannot post CONTEST transactions in time. In such scenarios, eventual consistency between blockchains is not guaranteed. As stated above, we use a block time of 13 s, therefore, we start our experiments with 10 s, and increase the period

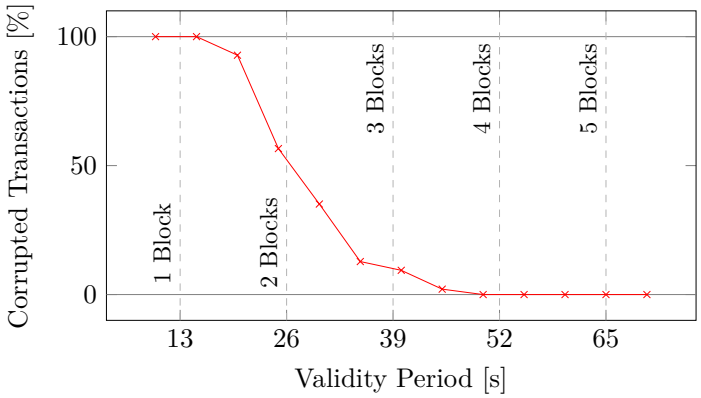

Figure 6.2: Impact of validity period on transaction success

by 5 s with each experiment. We then run our blockchain ecosystem for 30 min using each validity period and record the number of corrupted transactions. Note that we have to reset the inconsistent balances for wallets participating in a corrupted transaction in order to be able to run the experiments for 30 min.

Figure 6.2 shows the results of these experiments. Beyond 52 s, no corrupted transactions are observed. It becomes clear that using the reference implementation and waiting for 4 blocks (52 s) is sufficient for ensuring consistency. Between 1 and 3 blocks (13 s and 39 s, respectively), the amount of corrupted transactions declines with a varying rate.

From this experiment, we conclude that using a validity period with the length of at least 4 blocks (52 s) is sufficient to maintain consistency using our reference implementation. Additional time may be required in order to accommodate slow network connectivity.

6.3.2 Cost Analysis of DeXTT Transfers

To estimate the cost incurred by DeXTT transfers, we run the same experiment 20 times. Based on our previous experiment, we choose 65 s (5 blocks, well above the determined limit of 52 s) as the duration of the PoI validity period in each transaction. We record the average cost of each transaction. Table 6.5 shows an overview of the cost of the individual transactions involved in a DeXTT transfer. For each transaction, we show the mean cost, and its standard deviation, both in thousands of Ethereum Gas (kGas), and in United States Dollar (USD). For this, we assume a Gas price of 10 Gwei (1 Ether = 10^9 Gwei = 10^{18} wei) and a price of Ether of 115.71 USD.

These values were obtained from the Ethereum live chain in January 2019. Note that our implementation is optimized in that CLAIM and CONTEST both use the same smart contract function. Nevertheless, we distinguish the semantic

Table 6.5: Transaction cost analysis

Transaction	kGas (σ)	USD (σ)
CLAIM	57.7 (11.1)	0.0668 (0.0128)
CONTEST	81.5 (64.2)	0.0943 (0.0743)
FINALIZE	45.5 (0.1)	0.0527 (< 0.0001)
VETO	131.3 (91.9)	0.1520 (0.1063)
FINALIZE-VETO	48.6 (1.7)	0.0563 (0.0020)

difference (posting of new transfer for CLAIM, and participating in a contest for CONTEST) in the results.

In the following, we assume m blockchains and n total observers. For our calculation, we assume that all observers monitor all blockchains, and post CONTEST transactions if it benefits them. A regular DeXTT transfer (i.e., one which does not contain a conflicting PoI, and therefore requires no veto) consists of one CLAIM transaction (on the target chain), $\log_2 n$ CONTEST transactions (as discussed in Section 6.3.1) on each blockchain, i.e., $m \log_2 n$ CONTEST transactions, and m FINALIZE transactions. The CLAIM transaction is posted by the receiver, and each CONTEST transactions is posted by an observer (thus becoming a contestant). While the FINALIZE transaction can be posted by any party, posting it is beneficial to the receiver (because it finalizes the transfer to the receiver), and therefore it can be expected that the receiver will bear its cost to finalize the transfer.

The expected cost in kGas for a DeXTT transfer are as follows: The receiver bears the cost for one CLAIM transaction (57.7 kGas) and m FINALIZE transactions (45.5 kGas each). Each of the $\log_2 n$ expected observers posting transactions bears the cost for m CONTEST transactions (81.5 kGas each). The sender does not bear any cost.

Assuming a blockchain ecosystem of 10 blockchains, the total transaction cost for the receiver is 0.59 USD. Each of the $\log_2 n$ observers posting transactions bears cost of 0.94 USD. These numbers represent our current reference implementation and act as an upper bound for DeXTT transfer cost. Any additional optimization to the smart contract code has the potential to further reduce the Gas cost of the individual transactions, and therefore, of the overall DeXTT transfer. Additionally, these numbers allow us to reason about the economic impact of a currency using DeXTT transactions. Observers bear a transaction cost of 0.94 USD, and potentially receive a witness reward, currently defined as 1 CBT. The chance of an observer winning is $\frac{1}{n}$, however, according to the discussion in Section 6.3.1, on average, only $\log_2 n$ out of all n observers are expected to post CONTEST transactions. Therefore, the likelihood for an observer posting a transaction to win the contest is $\frac{\log_2 n}{n}$.

Therefore, the investment for each observer is 0.94 USD, the contest reward is 1 CBT, and the winning likelihood is $\frac{\log_2 n}{n}$. From this, it follows that in order for the observer to have incentive to post CONTEST transactions in an ecosystem of $m = 10$ blockchains, the inequation shown in (6.14) must hold, where p is the price of 1 CBT in USD.

$$\frac{\log_2 n}{n} p > 0.94 \text{ [USD]} \tag{6.14}$$

In other words, the price of 1 CBT in USD divided by the number of observers must be higher than 0.94. Assuming $n = 10$ observers, the CBT price must be above 2.83 USD. Assuming $n = 100$, the CBT price must be above 14.15 USD. For $n = 1000$, the CBT price must be above 94.32 USD. Ensuring this property is non-trivial, as the price of any asset is determined by supply and demand, and therefore the perceived value of CBT influences its price.

Note that these number assume $m = 10$ blockchains, and a fixed reward of 1 CBT. A pro rata reward, e.g., 1% of the transferred CBTs, would reduce the required CBT price, but also increase the complexity of calculating the witness incentive. Furthermore, a dynamic reward adaption based on the number of observers, similar to the variable mining rewards in Bitcoin, or a value selected by the sender, similar to the Gas price in Ethereum, can also be used to reduce the required CBT price, and therefore incentivize observers. We note again that these numbers pose an upper boundary for the expected DeXTT transfer cost and CBT price requirements for witness incentive.

6.4 Related Work

As discussed at the beginning of this chapter, cross-blockchain interoperability can be used to address the fragmentation of the blockchain research field. Yet, to the best of our knowledge, contemporary approaches provide only limited interoperability across blockchains.

Initial interoperability was limited to trading assets on centralized exchanges. Subsequently, decentralized exchanges such as Bisq [16] or 0x [260] emerged. Most recently, the Republic protocol [275] has been proposed, which includes a decentralized dark pool exchange, i.e., details about an exchange are kept secret.

All of these approaches, however, are concerned with the *exchange* of assets, generally using atomic swaps [123] for trustless asset exchange. In such an atomic swap, for instance, one party might transfer Bitcoin to another party, while the other party transfers Ether to the first. Therefore, each asset remains on its blockchain. In contrast, we propose a protocol for trading assets independently of a specific blockchains. In our approach, the balance information for such assets is stored on all blockchains simultaneously.

Another approach to creating a multi-blockchain framework is presented in PolkaDot [267]. PolkaDot aims to provide "the bedrock relay-chain" upon which data structures can be hosted. However, in contrast to our approach, no specifics about cross-blockchain asset transfers are provided. Instead, PolkaDot is explicitly not designed to be used as a currency [267]. Furthermore, PolkaDot is meant to be used as a basis for future blockchains (and other decentralized data structures), while in our current approach, we aim to use existing blockchains and implement functionality on top of them. However, the concepts presented in the PolkaDot paper are complementary to techniques we use in our approach.

Decentralized cross-blockchain transfers allow users to fully utilize the existing variety of blockchains, instead of being locked to a single blockchain. To the best of our knowledge, the approach closest to the work at hand is Metronome [196], which uses assets available on multiple blockchains. However, Metronome proposes that assets still lie on one specific blockchain at a time, while in our proposal, the assets are not bound to one blockchain.

The DeXTT protocol presented in this chapter is based on our own former work. The XPP has been formally described in [35]. Furthermore, in [36], we describe the deterministic witness selection approach conceptually. The work presented in this chapter significantly extends our former work by providing a concrete implementation of this approach within the DeXTT protocol. Furthermore, it changes aspects of our earlier work, in which we also defined a token existing across blockchains. Each wallet had a different balance on each blockchain (and all balances were recorded on all blockchains) [35]. In the work at hand, this concept is simplified, yielding only one balance per wallet (which is recorded on all blockchains).

6.5 Summary

In this chapter, we have presented the DeXTT protocol, which can be used for transferring CBTs. Within the DeXTT protocol, we face the problem of choosing a witness to receive a witness reward. Instead of aiming to predict the outcome of this contest on other blockchains from within a given blockchain, we solve it by designing the protocol such that this prediction is no longer necessary.

Using deterministic witnesses [36], we design the protocol in a way that defines the receiver of the witness reward in a deterministic and predictable way. In our evaluation, we show the feasibility of our approach, and perform a thorough cost analysis.

CHAPTER 7

Conclusions

The final chapter provides a summary of the main results achieved within this thesis. We revisit the research questions formulated in Section 1.2, critically analyze how these quesitons have been addressed by the contributions in Section 7.1, and give an overview of the contributions and their advancement of the scientific state of the art in Section 7.2. Finally, in Section 7.3, we give an overview of open questions and ongoing research, and provide potential ground for future work.

7.1 Research Questions Revisited

In Section 1.2, we have introduced three research questions. In this section, we discuss how our contributions address these questions in detail.

> **Research Question I.** How can predictive techniques be used in distributed systems to optimize operational cost, performance, and reliability?

In this thesis, we have shown various tools which can be used to realize predictive techniques in distributed systems. ANNs are a class of commonly used tools provided by the field of ML [181]. Numerous variations and extensions of ANNs exist. In Chapter 3, we use multi-layer ANNs to perform regression, which allows us to predict the resource utilization of tasks submitted to a cloud computing system. This can be used to optimize operational cost, since more cost-efficient placement, scheduling, and scaling decisions can be made based on a sufficiently precise prediction. In Chapter 4, we also use multi-layer ANNs, but in addition, employ techniques such as LSTM, convolutional, and recurrent layers in these ANNs. This allows us to use the ANN for classification at which step and

how likely a given process will fail. This can both decrease cost and increase reliability of the system, since predicting a failure allows to perform actions to mitigate or compensate this failure, e.g., sending another shipment of goods if the current shipment is likely to yield damaged goods.

> **Research Question II.** How can predictive approaches maintain functionality and performance in unknown and uncertain situations?

In situations where data is not precise (i.e., subject to noise), or entirely missing, system functionality and performance can be compromised. While Chapter 3 partially approaches such situations by using ANN regression to find such missing data, Chapter 5 is the main contribution addressing this question. In this contribution, we use EKF-based filtering to estimate and predict system state based on both intrinsic metrics (i.e., system state measurements) and extrinsic metrics (i.e., the system's environment). In our scenario, this is used to moderate the number of scaling operations, directly impacting the system's performance and cost.

> **Research Question III.** Can uncertainty be overcome by adapting the context of the system?

In Chapter 6, we show a scenario from the domain of blockchains where the recipient of a reward (the witness reward) must be the same on a number of blockchains. However, since direct communication between blockchains is not possible according to the XPP, we address this uncertainty by adapting the protocol using deterministic witnesses. While in contrast to the first two contributions, this solution implies a fundamental change in the system at hand, and assumes that such a change is possible, this contribution presents a feasible technique allowing us to deterministically define how such witnesses are selected, removing the uncertainty altogether.

7.2 Findings

In this thesis, we have shown how employing predictive approaches is beneficial in the domain of distributed systems. As use cases, we have shown several scenarios in different domains of distributed systems, and have showcased which methods can be used to meet various scenarios.

In Chapter 3, we have considered a cloud-based service provider and shown how ANNs can predict the resource utilization of tasks submitted by clients. This prediction can be used to proactively scale cloud resources owned by a

service provider. In this scenario, the goal of the prediction is to improve scaling and placement decisions of a distributed system. The dataset is labeled, and available in advance, hence, we can use offline supervised learning. The results of such learning can be directly used to improve the performance of a distributed system. In our case, the approach yields a prediction of resource utilization with 20% less prediction error compared to the baseline of linear regression.

In Chapter 4, an inter-organizational business process is used for international shipment of goods. We have used ANNs to predict the likelihood of a process step failing. Such a prediction can be used to mitigate failures, for instance, by ordering a new shipment of goods in case a damaged shipment becomes likely. In this scenario, the dataset is not available in advance, hence, we use online learning. Again, this dataset is labeled, since failures become evident—albeit after they occur—and we therefore again use supervised learning. The results of the learning performed in this contribution do not directly impact the system performance, but can be used to trigger mitigation mechanisms, and can therefore be used to improve the performance of the higher-order organizational unit. In our case, the approach yields a precision of 87% when predicting failures in a business process.

In Chapter 5, we have considered a research group analyzing images in a stream-like fashion. We have shown how to use EKFs to estimate and predict the load of the system used for image processing. This helps to moderate the number of scaling events, allowing to reduce cost stemming from scaling overhead. In this case, we do not have a labeled dataset, since the true state is unknown. Additionally, we employ an online estimation while the DSP system is operating. Using EKFs is beneficial in this scenarios since we can observe two quantities: On the one hand, we take measurements on the system state to be observed, on the other hand, we directly know the amount of data ingested by the DSP operator. This fits the EKF model well, where the system state is influenced both by itself (over time, using a state transition function), and by an input vector. EKFs also account for the noise found in both the system state measurements as well as the state transition function. In our case, EKFs allow a reduction of 88% in scaling events, compared to approaches not using EKFs.

Finally, in Chapter 6, we have visited the domain of blockchains and have shown how to use deterministic witnesses to approach a lack of predictability with systematic determinism. To record the transfer of value across various blockchains, we must offer a witness reward to nodes performing this transfer. However, the transfer would also have to be recorded across various blockchains, resulting in a recursive problem. We tackle this problem by introducing a deterministic process which is guaranteed to yield the same result (the same selection of a witness) on all blockchains. Such a solution is viable in scenarios where the prediction of a given data item is impossible, but the system itself (in our case, the protocol used for value transfer) can be altered.

7.3 Future Work

While the approaches presented in this thesis address the research questions as discussed in the previous section, in future work, several aspects can be investigated further.

First, additional data sources can be integrated into our solutions. While all of our predictive approaches are general enough to take into account additional data, doing so would naturally require additional evaluation. In Chapter 3, we use the task type and its input as our prediction data. Additional data includes the system state (e.g., its load), the client submitting the task, or context data such as the time of day. In Chapter 4, we use process and context events taken from two datasets. In Chapter 5, we use CPU load and memory utilization. Additional data such as I/O utilization or network traffic could be used.

In addition to extending the data sources, it would be interesting to investigate correlated data sources. For instance, in future work, we would like to investigate situations where higher network traffic correlates with higher I/O utilization, and the impact of such correlation on the proposed prediction techniques. Furthermore, alternative means of normalization of data such as cosine normalization [234] could be investigated. The impact of such normalization is particularly interesting in online learning scenarios, as used in Chapter 4.

In Chapter 3, we use a relatively simple ANN topology. In Chapter 4, this topology is extended using convolutional and recurrent layers. However, the effect of additional techniques such as LSTM or recurrent layers on the approach and scenario presented in Chapter 3 remains unexplored. With regard to the learning process itself, we also identify potential for optimization. We currently use Deeplearning4J, an existing third-party library, as a tool to perform our ML-based prediction without particular regard to optimization of runtime performance. In our setup, this is sufficient, since we assume a performant platform to execute the necessary computations. However, extending our work to devices with less computational power, such as those found in the IoT, would require optimizing the computation itself.

Finally, in our current application, we only consider single instances of the components taken into account in our system. In Chapter 3, we only consider one task a a time, in Chapter 4, we only consider a single business process, and in Chapter 5, we only consider a single DSP operator. However, in modern distributed systems, system-wide effects can occur, and their influence on the presented approaches remains to be investigated. For instance, in DSP, the system load of one operator could directly influence the amount of incoming data in a subsequent operator. Contemporary research is exploring this direction using decentralized approaches [51]. In this thesis, such ripple effects are not taken into account and therefore remain undiscovered.

Bibliography

[1] Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, and Mathias Weske. "Business process management: A survey". In: *International Conference on Business Process Management (BPM)*. LNCS 2678. Springer. 2003, pp. 1–12. DOI: 10.1007/3-540-44895-0_1.

[2] Daniel J. Abadi, Yanif Ahmad, Magdalena Balazinska, Uğur Çetintemel, Mitch Cherniack, Jeong-Hyon Hwang, Wolfgang Lindner, Anurag Maskey, Alex Rasin, Esther Ryvkina, Nesime Tatbul, Ying Xing, and Stan Zdonik. "The Design of the Borealis Stream Processing Engine." In: *Conference on Innovative Data Systems Research (CIDR)*. 2005, pp. 277–289.

[3] Asma Abu-Samah, Muhammad Kashif Shahzad, and Éric Zamaï. "Bayesian Based Methodology for the Extraction and Validation of Time Bound Failure Signatures for Online Failure Prediction". In: *Reliability Engineering & System Safety* 167 (2017), pp. 616–628. DOI: 10.1016/j.ress.2017.04.016.

[4] Lucio Agostinho, Guilherme Feliciano, Leonardo Olivi, Eleri Cardozo, and Eliane Guimarães. "A Bio-Inspired Approach to Provisioning of Virtual Resources in Federated Clouds". In: *9th IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC)*. IEEE, 2011, pp. 598–604. DOI: 10.1109/DASC.2011.109.

[5] Mohammed Alhamad, Tharam Dillon, and Elizabeth Chang. "SLA-Based Trust Model for Cloud Computing". In: *13th International Conference on Network-Based Information Systems (NBIS)*. IEEE. 2010, pp. 321–324. DOI: 10.1109/NBiS.2010.67.

[6] Rainer von Ammon. "Event-Driven Business Process Management". In: *Encyclopedia of Database Systems*. Ed. by Ling Liu and M. Tamer Özsu. Springer, 2009, pp. 1068–1071. DOI: 10.1007/978-0-387-39940-9_577.

[7] Rainer von Ammon, Christoph Emmersberger, Florian Springer, and Christian Wolff. "Event-Driven Business Process Management and its Practical Application Taking the Example of DHL". In: *1st International*

Workshop on Complex Event Processing for the Future Internet (iCEP). CEUR 412. 2008, article 8.

[8] Ashiq Anjum, Manu Sporny, and Alan Sill. "Blockchain Standards for Compliance and Trust". In: *IEEE Cloud Computing* 4.4 (2017), pp. 84–90. DOI: 10.1109/MCC.2017.3791019.

[9] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. "A View of Cloud Computing". In: *Communications of the ACM* 53.4 (2010), pp. 50–58. DOI: 10.1145/1721654.1721672.

[10] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. *Above the Clouds: A Berkeley View of Cloud Computing*. Tech. rep. UCB/EECS-2009-28. Electrical Engineering and Computer Sciences, University of California at Berkeley, 2009.

[11] Marcos Dias Assunção, Alexandre da Silva Veith, and Rajkumar Buyya. "Distributed Data Stream Processing and Edge Computing: A Survey on Resource Elasticity and Future Directions". In: *Journal of Network and Computer Applications* 103 (2018), pp. 1–17.

[12] Luigi Atzori, Antonio Iera, and Giacomo Morabito. "The Internet of Things: A Survey". In: *Computer Networks* 54.15 (2010), pp. 2787–2805. DOI: 10.1016/j.comnet.2010.05.010.

[13] Algirdas Avižienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. "Basic Concepts and Taxonomy of Dependable and Secure Computing". In: *IEEE Transactions on Dependable and Secure Computing* 1.1 (2004), pp. 11–33. DOI: 10.1109/TDSC.2004.2.

[14] Alistair Barros, Marlon Dumas, and Phillipa Oaks. "Standards for Web Service Choreography and Orchestration: Status and Perspectives". In: *Business Process Management Workshops (BPM)*. LNCS 3812. Springer, 2005, pp. 61–74. DOI: 10.1007/11678564_7.

[15] Christian Baun, Marcel Kunze, Jens Nimis, and Stefan Tai. *Cloud Computing: Web-Based Dynamic IT Services*. 2011. DOI: 10.1007/978-3-642-20917-8.

[16] Chris Beams. *Bisq - The Peer-to-Peer Bitcoin Exchange*. URL: https://github.com/bisq-network/bisq-docs/blob/master/exchange/whitepaper.adoc. White Paper. Version 2018-10-13. Accessed 2019-05-05.

[17] Anton Beloglazov and Rajkumar Buyya. “Energy Efficient Allocation of Virtual Machines in Cloud Data Centers”. In: *10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE. 2010, pp. 577–578. DOI: 10.1109/CCGRID.2010.45.

[18] Shai Ben-David, Eyal Kushilevitz, and Yishay Mansour. “Online Learning versus Offline Learning”. In: *Machine Learning* 29.1 (1997), pp. 45–63. DOI: 10.1023/A:1007465907571.

[19] Paul Razvan Berg and Mark Milton. *Chronos: An open protocol for streaming money*. 2018. URL: http://chronosprotocol.org/chronos-white-paper.pdf. White Paper. Version 0.2.2, 2018-08-17. Accessed 2019-05-05.

[20] David Bermbach, Frank Pallas, David García Pérez, Pierluigi Plebani, Maya Anderson, Ronen Kat, and Stefan Tai. “A Research Perspective on Fog Computing”. In: *Service-Oriented Computing - ICSOC 2017 Workshops*. Springer, 2018, pp. 198–210. DOI: 10.1007/978-3-319-91764-1_16.

[21] David Bermbach and Stefan Tai. “Benchmarking Eventual Consistency: Lessons Learned from Long-Term Experimental Studies”. In: *IEEE International Conference on Cloud Engineering (IC2E)*. IEEE. 2014, pp. 47–56. DOI: 10.1109/IC2E.2014.37.

[22] David Bermbach and Stefan Tai. “Eventual Consistency: How soon is eventual? An Evaluation of Amazon S3’s Consistency Behavior”. In: *6th Workshop for Middleware for Service-Oriented Computing (MW4SOC)*. IEEE. 2011, article 1. DOI: 10.1145/2093185.2093186.

[23] Fábio Bezerra, Jacques Wainer, and Wil M. P. van der Aalst. “Anomaly Detection Using Process Mining”. In: *Enterprise, Business-Process and Information Systems Modeling (BPMDS, EMMSAD)*. LNBIP 29. Springer, 2009, pp. 149–161. DOI: 10.1007/978-3-642-01862-6_13.

[24] Ofer Biran, Antonio Corradi, Mario Fanelli, Luca Foschini, Alexander Nus, Danny Raz, and Ezra Silvera. “A Stable Network-Aware VM Placement for Cloud Systems”. In: *12th International Conference on Cluster, Cloud and Grid Computing (CCGrid)*. IEEE. 2012, pp. 498–506. DOI: 10.1109/CCGrid.2012.119.

[25] Christopher Michael Bishop. *Neural Networks for Pattern Recognition*. Oxford, 1995. ISBN: 978-0-198-53849-3.

[26] Christopher Michael Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 2006. ISBN: 978-0-387-31073-2.

[27] Norman Bobroff, Andrzej Kochut, and Kirk Beaty. "Dynamic Placement of Virtual Machines for Managing SLA Violations". In: *10th IFIP/IEEE International Symposium on Integrated Network Management (INM)*. IEEE. 2007, pp. 119–128. DOI: 10.1109/INM.2007.374776.

[28] Kristof Böhmer and Stefanie Rinderle-Ma. "Multi-Perspective Anomaly Detection in Business Process Execution Events". In: *On the Move to Meaningful Internet Systems (OTM)*. LNCS 10033. Springer, 2016, pp. 80–98. DOI: 10.1007/978-3-319-48472-3_5.

[29] André Benjamin Bondi. "Characteristics of Scalability and their Impact on Performance". In: *2nd International Workshop on Software and Performance (WOSP)*. ACM. 2000, pp. 195–203. DOI: 10.1145/350391.350432.

[30] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. "Fog Computing and its Role in the Internet of Things". In: *1st Edition of the MCC Workshop on Mobile Cloud Computing (MCC)*. ACM. 2012, pp. 13–16. DOI: 10.1145/2342509.2342513.

[31] Michael Borkowski, Walid Fdhila, Matteo Nardelli, Stefanie Rinderle-Ma, and Stefan Schulte. "Event-Based Failure Prediction in Distributed Business Processes". In: *Information Systems* 81 (2019), pp. 220–235. DOI: 10.1016/j.is.2017.12.005.

[32] Michael Borkowski, Christoph Hochreiner, and Stefan Schulte. "Minimizing Cost by Reducing Scaling Operations in Distributed Stream Processing". In: *PVLDB* 12.7 (2019), pp. 724–737. DOI: 10.14778/3317315.3317316.

[33] Michael Borkowski, Christoph Hochreiner, and Stefan Schulte. "Moderated Resource Elasticity for Stream Processing Applications". In: *Parallel Processing Workshops (Euro-Par)*. LNCS 10659. Springer, 2017, pp. 5–16. DOI: 10.1007/978-3-319-75178-8_1.

[34] Michael Borkowski, Daniel McDonald, Christoph Ritzer, and Stefan Schulte. *Towards Atomic Cross-Chain Token Transfers: State of the Art and Open Questions within TAST*. 2018. DOI: 10.13140/RG.2.2.10769.48489. White Paper, TU Wien.

[35] Michael Borkowski, Christoph Ritzer, Daniel McDonald, and Stefan Schulte. *Caught in Chains: Claim-First Transactions for Cross-Blockchain Asset Transfers*. 2018. DOI: 10.13140/RG.2.2.24191.25769. White Paper, TU Wien.

[36] Michael Borkowski, Christoph Ritzer, and Stefan Schulte. *Deterministic Witnesses for Claim-First Transactions*. 2018. DOI: 10.13140/RG.2.2.17480.37123. White Paper, Technische Universität Wien.

[37] Michael Borkowski, Stefan Schulte, and Christoph Hochreiner. "Predicting Cloud Resource Utilization". In: *9th IEEE/ACM International Conference on Utility and Cloud Computing (UCC)*. IEEE/ACM, 2016, pp. 37–42. DOI: 10.1145/2996890.2996907.

[38] Michael Borkowski, Marten Sigwart, Philipp Frauenthaler, Taneli Hukkinen, and Stefan Schulte. "DeXTT: Deterministic Cross-Blockchain Token Transfers". In: *IEEE Access* 7.1 (2019), pp. 111030–111042. DOI: 10.1109/ACCESS.2019.2934707.

[39] Alessio Botta, Walter de Donato, Valerio Persico, and Antonio Pescapé. "Integration of Cloud Computing and Internet of Things: A Survey". In: *Future Generation Computer Systems* 56 (2016), pp. 684–700. DOI: 10.1016/j.future.2015.09.021.

[40] León Bottou. "Large-Scale Machine Learning with Stochastic Gradient Descent". In: *International Conference on Computational Statistics (COMPSTAT)*. Springer, 2010, pp. 177–186. DOI: 10.1007/978-3-7908-2604-3_16.

[41] Ivona Brandic. "Towards Self-manageable Cloud Services". In: *33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC)*. Vol. 2. IEEE. 2009, pp. 128–133. DOI: 10.1109/COMPSAC.2009.126.

[42] Ivona Brandic, Schahram Dustdar, Tobias Anstett, David Schumm, Frank Leymann, and Ralf Konrad. "Compliant Cloud Computing (C3): Architecture and Language Support for User-Driven Compliance Management in Clouds". In: *3rd IEEE International Conference on Cloud Computing (CLOUD)*. IEEE. 2010, pp. 244–251. DOI: 10.1109/CLOUD.2010.42.

[43] Ruth Breu, Schahram Dustdar, Johann Eder, Christian Huemer, Gerti Kappel, Julius Köpke, Philip Langer, Jürgen Mangler, Jan Mendling, Gustaf Neumann, Stefanie Rinderle-Ma, Stefan Schulte, Stefan Sobernig, and Barbara Weber. "Towards Living Inter-Organizational Processes". In: *15th IEEE Conference on Business Informatics (CBI)*. IEEE, 2013, pp. 363–366. DOI: 10.1109/CBI.2013.59.

[44] Alejandro P. Buchmann, Stefan Appel, Tobias Freudenreich, Sebastian Frischbier, and Pablo Ezequiel Guerrero. "From Calls to Events: Architecting Future BPM Systems". In: *International Conference on Business Process Management (BPM)*. LNCS 7481. Springer, 2012, pp. 17–32. DOI: 10.1007/978-3-642-32885-5_2.

[45] Rajkumar Buyya, Rajiv Ranjan, and Rodrigo Calheiros. "InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services". In: *International Conference on Algorithms*

and Architectures for Parallel Processing (ICA3PP). LNCS 6081. Springer. 2010, pp. 13–31. DOI: 10.1007/978-3-642-13119-6_2.

[46] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. "Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility". In: *Future Generation Computer Systems* 25.6 (2009), pp. 599–616. DOI: 10.1016/j.future.2008.12.001.

[47] Federico Cabitza, Raffaele Rasoini, and Gian Franco Gensini. "Unintended Consequences of Machine Learning in Medicine". In: *JAMA* 318.6 (2017), pp. 517–518. DOI: 10.1001/jama.2017.7797.

[48] Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposito, and Maria Luisa Villani. "QoS-Aware Replanning of Composite Web Services". In: *IEEE International Conference on Web Services (ICWS)*. IEEE, 2005, pp. 121–129. DOI: 10.1109/ICWS.2005.96.

[49] Valeria Cardellini, Emiliano Casalicchio, Vincenzo Grassi, Stefano Iannucci, Francesco Lo Presti, and Raffaela Mirandola. "Moses: A Framework for QoS Driven Runtime Adaptation of Service-Oriented Systems". In: *IEEE Transactions on Software Engineering* 38.5 (2012), pp. 1138–1159. DOI: 10.1109/TSE.2011.68.

[50] Valeria Cardellini, Emiliano Casalicchio, Francesco Lo Presti, and Luca Silvestri. "SLA-aware Resource Management for Application Service Providers in the Cloud". In: *First International Symposium on Network Cloud Computing and Applications (NCCA)*. IEEE. 2011, pp. 20–27. DOI: 10.1109/NCCA.2011.11.

[51] Valeria Cardellini, Mirko D'angelo, Vincenzo Grassi, Moreno Marzolla, and Raffaela Mirandola. "A Decentralized Approach to Network-Aware Service Composition". In: *European Conference on Service-Oriented and Cloud Computing (ESOCC)*. LNCS 9306. Springer. 2015, pp. 34–48. DOI: 10.1007/978-3-319-24072-5_3.

[52] Valeria Cardellini, Vincenzo Grassi, Francesco Lo Presti, and Matteo Nardelli. "Optimal Operator Placement for Distributed Stream Processing Applications". In: *10th ACM International Conference on Distributed and Event-based Systems (DEBS)*. ACM. 2016, pp. 69–80. DOI: 10.1145/2933267.2933312.

[53] Valeria Cardellini, Tihana Galinac Grbac, Matteo Nardelli, Nikola Tanković, and Hong-Linh Truong. "QoS-Based Elasticity for Service Chains in Distributed Edge Cloud Environments". In: *Autonomous Control for a Reliable Internet of Services*. Springer, 2018, pp. 182–211. DOI: 10.1007/978-3-319-90415-3_8.

[54] Valeria Cardellini, Francesco Lo Presti, Matteo Nardelli, and Gabriele Russo Russo. "Optimal Operator Deployment and Replication for Elastic Distributed Data Stream Processing". In: *Concurrency and Computation: Practice and Experience* 30.9 (2018), article e4334. DOI: 10.1002/cpe.4334.

[55] Valeria Cardellini, Matteo Nardelli, and Dario Luzi. "Elastic Stateful Stream Processing in Storm". In: *International Conference on High Performance Computing & Simulation (HPCS)*. IEEE. 2016, pp. 583–590. DOI: 10.1109/HPCSim.2016.7568388.

[56] Valeria Cardellini, Francesco Lo Presti, Matteo Nardelli, and Gabriele Russo Russo. "Auto-Scaling in Data Stream Processing Applications: A Model-Based Reinforcement Learning Approach". In: *Workshop on New Frontiers in Quantitative Methods in Informatics (InfQ)*. Springer. 2017, pp. 97–110. DOI: 10.1007/978-3-319-91632-3_8.

[57] Valeria Cardellini, Francesco Lo Presti, Matteo Nardelli, and Gabriele Russo Russo. "Decentralized Self-Adaptation for Elastic Data Stream Processing". In: *Future Generation Computer Systems* 87 (2018), pp. 171–185. DOI: 10.1016/j.future.2018.05.025.

[58] Eddy Caron, Frederic Desprez, and Adrian Muresan. "Forecasting for Grid and Cloud Computing On-Demand Resources Based on Pattern Matching". In: *2nd IEEE International Conference on Cloud Computing Technologies and Science (CloudCom)*. 2010, pp. 456–463. DOI: 10.1109/CloudCom.2010.65.

[59] Raul Castro Fernandez, Matteo Migliavacca, Evangelia Kalyvianaki, and Peter Pietzuch. "Integrating Scale Out and Fault Tolerance in Stream Processing Using Operator State Management". In: *ACM SIGMOD International Conference on Management of Data (SIGMOD)*. ACM. 2013, pp. 725–736. DOI: 10.1145/2463676.2465282.

[60] Sivadon Chaisiri, Bu-Sung Lee, and Dusit Niyato. "Optimization of Resource Provisioning Cost in Cloud Computing". In: *IEEE Transactions on Services Computing* 5.2 (2012), pp. 164–177. DOI: 10.1109/TSC.2011.7.

[61] Jeeva Chelladhurai, Pethuru Raj Chelliah, and Sathish Alampalayam Kumar. "Securing Docker Containers from Denial of Service (DoS) Attacks". In: *IEEE International Conference on Services Computing (SCC)*. IEEE. 2016, pp. 856–859. DOI: 10.1109/SCC.2016.123.

[62] Ming Chen, Hui Zhang, Ya-Yunn Su, Xiaorui Wang, Guofei Jiang, and Kenji Yoshihira. "Effective VM Sizing in Virtualized Data Centers".

In: *International Symposium on Integrated Network Management (IM)*. IEEE. 2011, pp. 594–601. DOI: 10.1109/INM.2011.5990564.

[63] Mitch Cherniack, Hari Balakrishnan, Magdalena Balazinska, Donald Carney, Ugur Cetintemel, Ying Xing, and Stanley B Zdonik. "Scalable Distributed Stream Processing." In: *First Biennial Conference on Innovative Data Systems Research (CIDR)*. Vol. 3. 2003, pp. 257–268.

[64] Ronan Collobert and Jason Weston. "A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning". In: *International Conference on Machine Learning (ICML)*. ACM, 2008, pp. 160–167. DOI: 10.1145/1390156.1390177.

[65] Raffaele Conforti, Marcello La Rosa, and Arthur H. M. ter Hofstede. "Filtering Out Infrequent Behavior from Business Process Event Logs". In: *IEEE Transactions on Knowledge and Data Engineering* 29.2 (2017), pp. 300–314. DOI: 10.1109/TKDE.2016.2614680.

[66] Raffaele Conforti, Massimiliano de Leoni, Marcello La Rosa, Wil M. P. van der Aalst, and Arthur H. M. ter Hofstede. "A Recommendation System for Predicting Risks Across Multiple Business Process Instances". In: *Decision Support Systems* 69 (2015), pp. 1–19. DOI: 10.1016/j.dss.2014.10.006.

[67] Georgiana Copil, Daniel Moldovan, Hong-Linh Truong, and Schahram Dustdar. "Multi-level Elasticity Control of Cloud Services". In: *International Conference on Service-Oriented Computing (ICSOC)*. LNCS 8274. Springer. 2013, pp. 429–436. DOI: 10.1007/978-3-642-45005-1_31.

[68] Antonio Corradi, Mario Fanelli, and Luca Foschini. "VM Consolidation: A Real Case Based on OpenStack Cloud". In: *Future Generation Computer Systems* 32 (2014), pp. 118–127. DOI: 10.1016/j.future.2012.05.012.

[69] Counterparty. *Counterparty*. URL: https://counterparty.io/docs/. Website. Accessed 2019-05-05.

[70] Counterparty. *Counterparty Protocol Specification*. 2018. URL: https://github.com/CounterpartyXCP/Documentation/blob/master/Developers/protocol_specification.md. Version 2018-10-02. Accessed 2019-05-05.

[71] Chris Dannen. *Introducing Ethereum and Solidity*. Springer, 2017. DOI: 10.1007/978-1-4842-2535-6.

[72] Sudipto Das, Shoji Nishimura, Divyakant Agrawal, and Amr El Abbadi. "Albatross: Lightweight Elasticity in Shared Storage Databases for the

Cloud Using Live Data Migration". In: *PVLDB* 4.8 (2011), pp. 494–505. DOI: 10.14778/2002974.2002977.

[73] Amir Vahid Dastjerdi and Rajkumar Buyya. "Fog Computing: Helping the Internet of Things Realize Its Potential". In: *Computer* 49.8 (2016), pp. 112–116. DOI: 10.1109/MC.2016.245.

[74] Christina Delimitrou and Christos Kozyrakis. "Quasar: Resource-Efficient and QoS-Aware Cluster Management". In: *19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM. 2014, pp. 127–144. DOI: 10.1145/2541940.2541941.

[75] Yahya Al-Dhuraibi, Fawaz Paraiso, Nabil Djarallah, and Philippe Merle. "Elasticity in Cloud Computing: State of the Art and Research Challenges". In: *IEEE Transactions on Services Computing* 11.2 (2017), pp. 430–447. DOI: 10.1109/TSC.2017.2711009.

[76] Robert Didden, Jeff Sigafoos, Mark F. O'Reilly, Giulio E. Lancioni, and Peter Sturmey. "A Multisite Cross-Cultural Replication of Unsuccessful Self-Treatment of Writer's Block". In: *Journal of Applied Behavior Analysis* 40.4 (2007), pp. 773–773. DOI: 10.1901/jaba.2007.773.

[77] Jeffrey Donahue, Lisa Anne Hendricks, Marcus Rohrbach, Subhashini Venugopalan, Sergio Guadarrama, Kate Saenko, and Trevor Darrell. "Long-Term Recurrent Convolutional Networks for Visual Recognition and Description". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.4 (2017), pp. 677–691. DOI: 10.1109/TPAMI.2016.2599174.

[78] Idilio Drago, Marco Mellia, Maurizio M. Munafo, Anna Sperotto, Ramin Sadre, and Aiko Pras. "Inside Dropbox: Understanding Personal Cloud Storage Services". In: *Internet Measurement Conference (IMC)*. ACM. 2012, pp. 481–494. DOI: 10.1145/2398776.2398827.

[79] Rajdeep Dua, A. Reddy Raja, and Dharmesh Kakadia. "Virtualization vs Containerization to Support PaaS". In: *IEEE International Conference on Cloud Engineering (IC2E)*. IEEE. 2014, pp. 610–614. DOI: 10.1109/IC2E.2014.41.

[80] Thang Le Duc, Rafael García Leiva, Paolo Casari, and Per-Olov Östberg. "Machine Learning Methods for Reliable Resource Provisioning in Edge-Cloud Computing: A Survey". In: *ACM Computing Surveys* 52.5 (2019), p. 94. DOI: 10.1145/3341145.

[81] Schahram Dustdar, Yike Guo, Benjamin Satzger, and Hong-Linh Truong. "Principles of Elastic Processes". In: *IEEE Internet Computing* 15.5 (2011), pp. 66–71. DOI: 10.1109/MIC.2011.121.

[82] Xavier Dutreilh, Sergey Kirgizov, Olga Melekhova, Jacques Malenfant, Nicolas Rivierre, and Isis Truck. "Using Reinforcement Learning for Autonomic Resource Allocation in Clouds: Towards a Fully Automated Workflow". In: *7th International Conference on Autonomic and Autonomous Systems (ICAS)*. IARIA. 2011, pp. 67–74.

[83] Jacob Eberhardt and Stefan Tai. "On or Off the Blockchain? Insights on Off-Chaining Computation and Data". In: *European Conference on Service-Oriented and Cloud Computing (ESOCC)*. LNCS 10465. Springer. 2017, pp. 3–15. DOI: 10.1007/978-3-319-67262-5_1.

[84] Jacob Eberhardt and Stefan Tai. "ZoKrates - Scalable Privacy-Preserving Off-Chain Computations". In: *IEEE International Conference on Blockchain*. 2018, pp. 1084–1091. DOI: 10.1109/Cybermatics_2018.2018.00199.

[85] Dmitry Efanov and Pavel Roschin. "The All-Pervasiveness of the Blockchain Technology". In: *Procedia Computer Science* 123 (2018), pp. 116–121. DOI: 10.1016/j.procs.2018.01.019.

[86] Dominik Ernst, David Bermbach, and Stefan Tai. "Understanding the container ecosystem: A taxonomy of building blocks for container lifecycle and cluster management". In: *IEEE Second International Workshop on Container Technologies and Container Clouds (WoC)*. IEEE, 2016.

[87] Opher Etzion and Peter Niblett. *Event Processing in Action*. Manning Publications, 2010. ISBN: 978-1-935-18221-4.

[88] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. "The Many Faces of Publish/Subscribe". In: *ACM Computing Surveys* 35.2 (2003), pp. 114–131. DOI: 10.1145/857076.857078.

[89] Seven Euting, Christian Janiesch, Robin Fischer, Stefan Tai, and Ingo Weber. "Scalable Business Process Execution in the Cloud". In: *IEEE International Conference on Cloud Engineering (IC2E)*. IEEE. 2014, pp. 175–184. DOI: 10.1109/IC2E.2014.13.

[90] Joerg Evermann, Jana-Rebecca Rehse, and Peter Fettke. "Predicting Process Behaviour Using Deep Learning". In: *Decision Support Systems* 100 (2017), pp. 129–140. DOI: 10.1016/j.dss.2017.04.003.

[91] Guisheng Fan, Huiqun Yu, Liqiong Chen, and Dongmei Liu. "A Petri Net-Based Byzantine Fault Diagnosis Method for Service Composition". In: *36th IEEE Computer Software and Applications Conference (COMPSAC)*. IEEE, 2012, pp. 42–51. DOI: 10.1109/COMPSAC.2012.63.

[92] Walid Fdhila, Conrad Indiono, Stefanie Rinderle-Ma, and Manfred Reichert. "Dealing with Change in Process Choreographies: Design and

Implementation of Propagation Algorithms". In: *Information Systems* 49 (2015), pp. 1–24. DOI: 10.1016/j.is.2014.10.004.

[93] Walid Fdhila and Stefanie Rinderle-Ma. "Predicting Change Propagation Impacts in Collaborative Business Processes". In: *29th Annual ACM Symposium on Applied Computing (SAC)*. ACM. 2014, pp. 1378–1385. DOI: 10.1145/2554850.2554966.

[94] Walid Fdhila, Stefanie Rinderle-Ma, Aymen Baouab, Olivier Perrin, and Claude Godart. "On Evolving Partitioned Web Service Orchestrations". In: *5th IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*. 2012, pp. 1–6. DOI: 10.1109/SOCA.2012.6449446.

[95] Walid Fdhila, Stefanie Rinderle-Ma, David Knuplesch, and Manfred Reichert. "Change and Compliance in Collaborative Processes". In: *IEEE International Conference on Services Computing (SCC)*. IEEE, 2015, pp. 162–169. DOI: 10.1109/SCC.2015.31.

[96] Walid Fdhila, Stefanie Rinderle-Ma, and Manfred Reichert. "Change Propagation in Collaborative Processes Scenarios". In: *8th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*. IEEE. 2012, pp. 452–461. DOI: 10.4108/icst.collaboratecom.2012.250408.

[97] Zohar Feldman, Fabiana Fournier, Rod Franklin, and Andreas Metzger. "Proactive Event Processing in Action: A Case Study on the Proactive Management of Transport Processes (Industry Article)". In: *7th ACM International Conference on Distributed Event-Based Systems (DEBS)*. ACM, 2013, pp. 97–106. DOI: 10.1145/2488222.2488274.

[98] Avrilia Floratou, Ashvin Agrawal, Bill Graham, Sriram Rao, and Karthik Ramasamy. "Dhalion: Self-Regulating Stream Processing in Heron". In: *PVLDB* 10.12 (2017), pp. 1825–1836. DOI: 10.14778/3137765.3137786.

[99] Francesco Folino, Massimo Guarascio, and Luigi Pontieri. "Discovering Context-Aware Models for Predicting Business Process Performances". In: *On the Move to Meaningful Internet Systems (OTM)*. LNCS 7565. Springer. 2012, pp. 287–304. DOI: 10.1007/978-3-642-33606-5_18.

[100] Jerome H. Friedman. "Data Mining and Statistics: What's the Connection?" In: *29th Symposium on the Interface Between Computer Science and Statistics*. Springer. 1998, pp. 1–7. DOI: 10.1007/978-1-4612-2856-1.

[101] Anshul Gandhi, Yuan Chen, Daniel Gmach, Martin Arlitt, and Manish Marwah. "Hybrid Resource Provisioning for Minimizing Data Center SLA Violations and Power Consumption". In: *Sustainable Computing: Informatics and Systems* 2.2 (2012), pp. 91–104. DOI: 10.1016/j.suscom.2012.01.005.

[102] Anshul Gandhi, Parijat Dube, Alexei Karve, Andrzej Kochut, and Li Zhang. "Model-Driven Optimal Resource Scaling in Cloud". In: *Software & Systems Modeling* 17.2 (2018), pp. 509–526. DOI: 10.1007/s10270-017-0584-y.

[103] Atefeh Gholipour and Sattar Mirzakuchaki. "A Pseudorandom Number Generator with KECCAK Hash Function". In: *International Journal of Computer and Electrical Engineering* 3.6 (2011), pp. 896–899. DOI: 10.7763/IJCEE.2011.V3.439.

[104] Henri Gilbert and Helena Handschuh. "Security Analysis of SHA-256 and Sisters". In: *International Workshop on Selected Areas in Cryptography (SAC)*. Springer. 2003, pp. 175–193. DOI: 10.1007/978-3-540-24654-1_13.

[105] Florian Glaser and Luis Bezzenberger. "Beyond Cryptocurrencies – A Taxonomy of Decentralized Consensus Systems". In: *23rd European Conference on Information Systems (ECIS)*. 2015, paper 57, 18 pages. DOI: 10.18151/7217326.

[106] Xavier Glorot and Yoshua Bengio. "Understanding the Difficulty of Training Deep Feedforward Neural Networks". In: *13th International Conference on Artificial Intelligence and Statistics (AISTATS)*. JMLR 9. PMLR, 2010, pp. 249–256. DOI: 10.1.1.207.2059.

[107] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep sparse rectifier neural networks". In: *14th International Conference on Artificial Intelligence and Statistics (AISTATS)*. JMLR 15. PMLR, 2011, pp. 315–323. DOI: 10.1.1.208.6449.

[108] Zhenhuan Gong, Xiaohui Gu, and John Wilkes. "PRESS: Predictive Elastic Resource Scaling for Cloud Systems". In: *International Conference on Network and Service Management (CNSM)*. IEEE. 2010, pp. 9–16. DOI: 10.1109/CNSM.2010.5691343.

[109] Graham C. Goodwin, Stefan F. Graebe, and Mario E. Salgado. *Control System Design*. Prentice Hall, 2001. ISBN: 978-0-139-58653-8.

[110] Louis Gosselin, Maxime Tye-Gingras, and François Mathieu-Potvin. "Review of Utilization of Genetic Algorithms in Heat Transfer Problems". In: *International Journal of Heat and Mass Transfer* 52.9-10 (2009), pp. 2169–2188. DOI: 10.1016/j.ijheatmasstransfer.2008.11.015.

[111] Raúl Gracia-Tinedo, Yongchao Tian, Josep Sampé, Hamza Harkous, John Lenton, Pedro García-López, Marc Sánchez-Artigas, and Marko Vukolic. "Dissecting UbuntuOne: Autopsy of a Global-Scale Personal Cloud Back-End". In: *Internet Measurement Conference (IMC)*. ACM. 2015, pp. 155–168. DOI: 10.1145/2815675.2815677.

[112] Gregor Grambow, Roy Oberhauser, and Manfred Reichert. "Event-Driven Exception Handling for Software Engineering Processes". In: *International Conference on Business Process Management (BPM)*. LNBIP 99. 2012, pp. 414–426. DOI: 10.1007/978-3-642-28108-2_40.

[113] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. "Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions". In: *Future Generation Computer Systems* 29.7 (2013), pp. 1645–1660. DOI: 10.1016/j.future.2013.01.010.

[114] Vincenzo Gulisano, Ricardo Jimenez-Peris, Marta Patino-Martinez, Claudio Soriente, and Patrick Valduriez. "StreamCloud: An Elastic and Scalable Data Streaming System". In: *IEEE Transactions on Parallel and Distributed Systems* 23.12 (2012), pp. 2351–2365. DOI: 10.1109/TPDS.2012.24.

[115] Richard H. R. Hahnloser, Rahul Sarpeshkar, Misha A. Mahowald, Rodney J. Douglas, and H. Sebastian Seung. "Digital Selection and Analogue Amplification Coexist in a Cortex-Inspired Silicon Circuit". In: *Nature* 405.6789 (2000), pp. 947–951. DOI: 10.1038/35016072.

[116] Rui Han, Li Guo, Moustafa M. Ghanem, and Yike Guo. "Lightweight Resource Scaling for Cloud Applications". In: *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 2012, pp. 644–651. DOI: 10.1109/CCGrid.2012.52.

[117] Edward James Hannan. *Multiple Time Series*. Wiley, 1970. ISBN: 978-0-471-34805-4. DOI: 10.1002/9780470316429.

[118] Fang Hao, Murali Kodialam, T. V. Lakshman, and Sarit Mukherjee. "Online Allocation of Virtual Machines in a Distributed Cloud". In: *IEEE/ACM Transactions on Networking* 25.1 (2017), pp. 238–249. DOI: 10.1109/TNET.2016.2575779.

[119] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. 2nd Edition. Prentice Hall, 1998. ISBN: 978-0-132-73350-2.

[120] Robert Hecht-Nielsen. "Theory of the Backpropagation Neural Network". In: *Neural Networks for Perception: Computation, Learning, and Architectures*. 1992, pp. 65–93. DOI: 10.1016/B978-0-12-741252-8.50010-8.

[121] Thomas Heinze, Lars Roediger, Andreas Meister, Yuanzhen Ji, Zbigniew Jerzak, and Christof Fetzer. "Online Parameter Optimization for Elastic Data Stream Processing". In: *6th ACM Symposium on Cloud Computing (SoCC)*. ACM, 2015, pp. 276–287. DOI: 10.1145/2806777.2806847.

[122] Nikolas Roman Herbst, Samuel Kounev, and Ralf H Reussner. "Elasticity in Cloud Computing: What It Is, and What It Is Not". In: *10th International Conference on Autonomic Computing (ICAC)*. Vol. 13. USENIX. 2013, pp. 23–27.

[123] Maurice Herlihy. "Atomic Cross-Chain Swaps". In: *ACM Symposium on Principles of Distributed Computing (PODC)*. ACM, 2018, pp. 245–254. DOI: 10.1145/3212734.3212736.

[124] Gabriel Hermosillo, Lionel Seinturier, and Laurence Duchien. "Creating Context-Adaptive Business Processes". In: *International Conference on Service-Oriented Computing (ICSOC)*. LNCS 6470. Springer, 2010, pp. 228–242. DOI: 10.1007/978-3-642-17358-5_16.

[125] Sergio Hernández, Sebastiaan J. van Zelst, Joaquín Ezpeleta, and Wil M. P. van der Aalst. "Handling Big(ger) Logs: Connecting ProM 6 to Apache Hadoop". In: *Proceedings of the BPM Demo Session 2015*. CEUR Workshop Proceedings 1418. CEUR, 2015, pp. 80–84.

[126] Jordi Herrera-Joancomartí and Cristina Pérez-Solà. "Privacy in Bitcoin Transactions: New Challenges from Blockchain Scalability Solutions". In: *Modeling Decisions for Artificial Intelligence (MDAI)*. Springer, 2016, pp. 26–44. DOI: 10.1007/978-3-319-45656-0_3.

[127] Michael G. Hinchey and Roy Sterritt. "Self-Managing Software". In: *Computer* 39.2 (2006), pp. 107–109. DOI: 10.1109/MC.2006.69.

[128] Yoichi Hirai. "Defining the Ethereum Virtual Machine for Interactive Theorem Provers". In: *International Conference on Financial Cryptography and Data Security (FC)*. Springer, 2017, pp. 520–535. DOI: 10.1007/978-3-319-70278-0_33.

[129] Christoph Hochreiner, Stefan Schulte, Schahram Dustdar, and Freddy Lecue. "Elastic Stream Processing for Distributed Environments". In: *Internet Computing* 19.6 (2015), pp. 54–59. DOI: 10.1109/MIC.2015.118.

[130] Christoph Hochreiner, Michael Vögler, Stefan Schulte, and Schahram Dustdar. "Cost-Efficient Enactment of Stream Processing Topologies". In: *PeerJ Computer Science* 3 (2017), article e141. DOI: 10.7717/peerj-cs.141.

[131] Christoph Hochreiner, Michael Vögler, Stefan Schulte, and Schahram Dustdar. "Elastic Stream Processing for the Internet of Things". In: *9th IEEE International Conference on Cloud Computing (CLOUD)*. IEEE, 2016, pp. 100–107. DOI: 10.1109/CLOUD.2016.0023.

[132] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

[133] Philipp Hoenisch, Christoph Hochreiner, Dieter Schuller, Stefan Schulte, Jan Mendling, and Schahram Dustdar. "Cost-Efficient Scheduling of Elastic Processes in Hybrid Clouds". In: *8th IEEE International Conference on Cloud Computing (CLOUD)*. IEEE, 2015, pp. 17–24. DOI: 10.1109/CLOUD.2015.13.

[134] Philipp Hoenisch, Dieter Schuller, Stefan Schulte, Christoph Hochreiner, and Schahram Dustdar. "Optimization of Complex Elastic Processes". In: *IEEE Transactions on Services Computing* 9.5 (2016), pp. 700–713. DOI: 10.1109/TSC.2015.2428246.

[135] Philipp Hoenisch, Stefan Schulte, Schahram Dustdar, and Srikumar Venugopal. "Self-Adaptive Resource Allocation for Elastic Process Execution". In: *6th IEEE International Conference on Cloud Computing (CLOUD)*. IEEE, 2013, pp. 220–227. DOI: 10.1109/CLOUD.2013.126.

[136] Philipp Hoenisch, Ingo Weber, Stefan Schulte, Liming Zhu, and Alan Fekete. "Four-Fold Auto-Scaling on a Contemporary Deployment Platform Using Docker Containers". In: *International Conference on Service-Oriented Computing (ICSOC)*. LNCS 9435. Springer, 2015, pp. 316–323. DOI: 10.1007/978-3-662-48616-0_20.

[137] Pablo Hofbauer, Jangwook P. Jung, Tanner J. McArdle, and Brenda M. Ogle. "Simple Monolayer Differentiation of Murine Cardiomyocytes via Nutrient Deprivation-Mediated Activation of β-Catenin". In: *Stem Cell Reviews and Reports* 12.6 (2016), pp. 731–743. DOI: 10.1007/s12015-016-9678-0.

[138] Gao Huang, Shiji Song, Jatinder N. D. Gupta, and Cheng Wu. "Semi-Supervised and Unsupervised Extreme Learning Machines". In: *IEEE Transactions on Cybernetics* 44.12 (2014), pp. 2405–2417. DOI: 10.1109/TCYB.2014.2307349.

[139] IEEE. *IEEE Standard for eXtensible Event Stream (XES) for Achieving Interoperability in Event Logs and Event Streams*. 2016. DOI: 10.1109/IEEESTD.2016.7740858. IEEE Standard 1849-2016.

[140] Sadeka Islam, Jacky Keung, Kevin Lee, and Anna Liu. "Empirical Prediction Models for Adaptive Resource Provisioning in the Cloud". In:

Future Generation Computer Systems 28.1 (2012), pp. 155–162. DOI: 10.1016/j.future.2011.05.027.

[141] Raj Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling.* Wiley, 1991. ISBN: 978-0-471-50336-1.

[142] Christian Janiesch, Martin Matzner, and Oliver Müller. "Beyond Process Monitoring: A Proof-of-Concept of Event-Driven Business Activity Management". In: *Business Process Management Journal* 18.4 (2012), pp. 625–643. DOI: 10.1108/14637151211253765.

[143] Andrew H. Jazwinski. *Stochastic Processes and Filtering Theory.* Dover, 2007. ISBN: 978-0-486-46274-5.

[144] Martin Jergler, Mohammad Sadoghi, and Hans-Arno Jacobsen. "Geo-Distribution of Flexible Business Processes over Publish/Subscribe Paradigm". In: *17th International Middleware Conference.* ACM, 2016, article 15. DOI: 10.1145/2988336.2988351.

[145] Dejun Jiang, Guillaume Pierre, and Chi-Hung Chi. "Autonomous Resource Provisioning for Multi-Service Web Applications". In: *19th International Conference on World Wide Web (WWW).* ACM. 2010, pp. 471–480. DOI: 10.1145/1772690.1772739.

[146] Jing Jiang, Jie Lu, Guangquan Zhang, and Guodong Long. "Optimal Cloud Resource Auto-Scaling for Web Applications". In: *13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (CCGrid)* (2013), pp. 58–65. DOI: 10.1109/CCGrid.2013.73.

[147] Thorsten Joachims. "Text Categorization With Support Vector Machines: Learning with Many Relevant Features". In: *European Conference on Machine Learning (ECML).* LNCS 1398. Springer. 1998, pp. 137–142. DOI: 10.1007/BFb0026683.

[148] Prasad Jogalekar and Murray Woodside. "Evaluating the Scalability of Distributed Systems". In: *IEEE Transactions on Parallel and Distributed Systems* 11.6 (2000), pp. 589–603. DOI: 10.1109/71.862209.

[149] Don Johnson, Alfred Menezes, and Scott Vanstone. "The Elliptic Curve Digital Signature Algorithm". In: *International Journal of Information Security* 1.1 (2001), pp. 36–63. DOI: 10.1007/s102070100002.

[150] Sandra Johnson, Peter Robinson, and John Brainard. "Sidechains and Interoperability". In: *CoRR* arXiv:1903.04077 (2019).

[151] Rudolph E. Kalman and Richard S. Bucy. "New Results in Linear Filtering and Prediction Theory". In: *Journal of Basic Engineering* 83.1 (1961), pp. 95–108. DOI: 10.1115/1.3658902.

[152] Bokyoung Kang, Dongsoo Kim, and Suk-Ho Kang. "Real-Time Business Process Monitoring Method for Prediction of Abnormal Termination using KNNI-Based LOF Prediction". In: *Expert Systems with Applications* 39.5 (2012), pp. 6061–6068. DOI: 10.1016/j.eswa.2011.12.007.

[153] AV Karthick, E Ramaraj, and R Ganapathy Subramanian. "An Efficient Multi Queue Job Scheduling for Cloud Computing". In: *World Congress on Computing and Communication Technologies (WCCCT)*. IEEE. 2014, pp. 164–166. DOI: 10.1109/WCCCT.2014.8.

[154] Arun Kejariwal and John Allspaw. *The Art of Capacity Planning: Scaling Web Resources in the Cloud*. O'Reilly, 2017. ISBN: 978-1-491-93920-8.

[155] Jeffrey O. Kephart and David M. Chess. "The Vision of Autonomic Computing". In: *Computer* 36.1 (2003), pp. 41–50. DOI: 10.1109/MC.2003.1160055.

[156] Arijit Khan, Xifeng Yan, Shu Tao, and Nikos Anerousis. "Workload Characterization and Prediction in the Cloud: A Multiple Time Series Approach". In: *IEEE Network Operations and Management Symposium*. IEEE. 2012, pp. 1287–1294. DOI: 10.1109/NOMS.2012.6212065.

[157] Jonghoon Kim, Jisu Hong, and Hyunjin Park. "Prospects of Deep Learning for Medical Imaging". In: *Precision and Future Medicine* 2.2 (2018), pp. 37–52. DOI: 10.23838/pfm.2018.00030.

[158] Markus Klems, Jacob Eberhardt, Stefan Tai, Steffen Härtlein, Simon Buchholz, and Ahmed Tidjani. "Trustless Intermediation in Blockchain-Based Decentralized Service Marketplaces". In: *International Conference on Service-Oriented Computing (ICSOC)*. Springer, 2017, pp. 731–739. DOI: 10.1007/978-3-319-69035-3_53.

[159] David Knuplesch, Manfred Reichert, Rüdiger Pryss, Walid Fdhila, and Stefanie Rinderle-Ma. "Ensuring Compliance of Distributed and Collaborative Workflows". In: *9th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*. IEEE. 2013, pp. 133–142. DOI: 10.4108/icst.collaboratecom.2013.254095.

[160] Falko Koetter and Monika Kochanowski. "A Model-Driven Approach for Event-based Business Process Monitoring". In: *Information Systems and e-Business Management* 13.1 (2015), pp. 5–36. DOI: 10.1007/s10257-014-0233-8.

[161] Ron Kohavi. "A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection". In: *International Joint Conference on Artificial Intelligence (IJCAI)*. Morgan Kaufmann. 1995, pp. 1137–1143. ISBN: 1-55860-363-8.

[162] Jaehyun Kong, Jae-Yoon Jung, and Jinwoo Park. "Event-Driven Service Coordination for Business Process Integration in Ubiquitous Enterprises". In: *Computers & Industrial Engineering* 57.1 (2009), pp. 14–26. DOI: 10.1016/j.cie.2008.08.019.

[163] Madhukar Korupolu, Aameek Singh, and Bhuvan Bamba. "Coupled Placement in Modern Data Centers". In: *IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*. IEEE. 2009, pp. 1–12. DOI: 10.1109/IPDPS.2009.5161067.

[164] Julian Krumeich, Benjamin Weis, Dirk Werth, and Peter Loos. "Event-Driven Business Process Management: Where are we Now?" In: *Business Process Management Journal* 20.4 (2014), pp. 615–633. DOI: 10.1108/BPMJ-07-2013-0092.

[165] Karthik Kumar and Yung-Hsiang Lu. "Cloud Computing for Mobile Users: Can Offloading Computation Save Energy?" In: *Computer* 43.4 (2010), pp. 51–56. DOI: 10.1109/MC.2010.98.

[166] Ewnetu Bayuh Lakew, Cristian Klein, Francisco Hernandez-Rodriguez, and Erik Elmroth. "Towards Faster Response Time Models for Vertical Elasticity". In: *IEEE/ACM 7th International Conference on Utility and Cloud Computing (UCC)*. IEEE. 2014, pp. 560–565. DOI: 10.1109/UCC.2014.86.

[167] Pat Langley. *Elements of Machine Learning*. Morgan Kaufmann, 1996. ISBN: 1-55860-301-8.

[168] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep Learning". In: *Nature* 521.7553 (2015), pp. 436–444. DOI: 10.1038/nature14539.

[169] Sander J. J. Leemans, Dirk Fahland, and Wil M. P. van der Aalst. "Discovering Block-Structured Process Models from Event Logs – A Constructive Approach". In: *International Conference on Application and Theory of Petri Nets and Concurrency (PETRI NETS)*. Springer, 2013, pp. 311–329. DOI: 10.1007/978-3-642-38697-8_17.

[170] Philipp Leitner, Waldemar Hummer, and Schahram Dustdar. "Cost-Based Optimization of Service Compositions". In: *IEEE Transactions on Services Computing* 6.2 (2013), pp. 239–251. DOI: 10.1109/TSC.2011.53.

[171] Philipp Leitner, Waldemar Hummer, Benjamin Satzger, Christian Inzinger, and Schahram Dustdar. "Cost-Efficient and Application SLA-Aware Client Side Request Scheduling in an Infrastructure-as-a-Service Cloud". In: *5th IEEE International Conference on Cloud Computing (CLOUD)*. IEEE, 2012, pp. 213–220. DOI: 10.1109/CLOUD.2012.21.

[172] Alexander Lenk, Markus Klems, Jens Nimis, Stefan Tai, and Thomas Sandholm. "What's Inside the Cloud? An Architectural Map of the Cloud Landscape". In: *ICSE Workshop on Software Engineering Challenges of Cloud Computing*. IEEE. 2009, pp. 23–31. DOI: 10.1109/CLOUD.2009.5071529.

[173] Anna Leontjeva, Raffaele Conforti, Chiara Di Francescomarino, Marlon Dumas, and Fabrizio Maria Maggi. "Complex Symbolic Sequence Encodings for Predictive Monitoring of Business Processes". In: *International Conference on Business Process Management (BPM)*. Springer, 2015, pp. 297–313. DOI: 10.1007/978-3-319-23063-4_21.

[174] Guoli Li, Vinod Muthusamy, and Hans-Arno Jacobsen. "A Distributed Service-oriented Architecture for Business Process Execution". In: *ACM Transactions on the Web* 4.1 (2010), article 2. DOI: 10.1145/1658373.1658375.

[175] Jimmy Lin and Alek Kolcz. "Large-Scale Machine Learning at Twitter". In: *ACM SIGMOD International Conference on Management of Data (SIGMOD)*. ACM. 2012, pp. 793–804. DOI: 10.1145/2213836.2213958.

[176] Tony Lindeberg. "Scale-Space for Discrete Signals". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 12.3 (1990), pp. 234–254. DOI: 10.1109/34.49051.

[177] Fabio Lopez-Pires and Benjamin Baran. "Virtual Machine Placement Literature Review". In: *CoRR* arXiv:1506.01509 (2015).

[178] Tania Lorido-Botran, José Miguel-Alonso, and Jose Antonio Lozano. "A Review of Auto-Scaling Techniques for Elastic Applications in Cloud Environments". In: *Journal of Grid Computing* 12.4 (2014), pp. 559–592. DOI: https://doi.org/10.1007/s10723-014-9314-7.

[179] David Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley, 2002. ISBN: 978-0-201-72789-0.

[180] Kasper Grud Skat Madsen, Yongluan Zhou, and Jianneng Cao. "Integrative Dynamic Reconfiguration in a Parallel Stream Processing Engine". In: *33rd IEEE International Conference on Data Engineering (ICDE)*. IEEE. 2017, pp. 227–230. DOI: 10.1109/ICDE.2017.81.

[181] Carolyn Mair, Gada Kadoda, Martin Lefley, Keith Phalp, Chris Schofield, Martin Shepperd, and Steve Webster. "An Investigation of Machine Learning Based Prediction Systems". In: *Journal of Systems and Software* 53.1 (2000), pp. 23–29. DOI: 10.1016/S0164-1212(00)00005-4.

[182] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008. ISBN: 978-0521865715.

[183] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999. ISBN: 978-0-262-13360-9.

[184] Ronny S Mans, M. H. Schonenberg, Minseok Song, Wil M. P. van der Aalst, and Piet J. M. Bakker. "Application of Process Mining in Healthcare – A Case Study in a Dutch Hospital". In: *International Joint Conference on Biomedical Engineering Systems and Technologies (BIOSTEC)*. CCIS 25. Springer. 2008, pp. 425–438. DOI: 10.1007/978-3-540-92219-3_32.

[185] Ming Mao and Marty Humphrey. "Auto-Scaling to Minimize Cost and Meet Application Deadlines in Cloud Workflows". In: *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. IEEE. 2011, article 49. DOI: 10.1145/2063384.2063449.

[186] Karl Mason, Martin Duggan, Enda Barrett, Jim Duggan, and Enda Howley. "Predicting Host CPU Utilization in the Cloud using Evolutionary Neural Networks". In: *Future Generation Computer Systems* 86 (2018), pp. 162–173. DOI: 10.1016/j.future.2018.03.040.

[187] Masakazu Matsugu, Katsuhiko Mori, Yusuke Mitari, and Yuji Kaneda. "Subject Independent Facial Expression Recognition with Robust Face Detection using a Convolutional Neural Network". In: *Neural Networks* 16.5 (2003), pp. 555–559. DOI: 10.1016/S0893-6080(03)00115-1.

[188] Brian W. Matthews. "Comparison of the Predicted and Observed Secondary Structure of T4 Phage Lysozyme". In: *Biochimica et Biophysica Acta (BBA) – Protein Structure* 405.2 (1975), pp. 442–451. DOI: 10.1016/0005-2795(75)90109-9.

[189] Michael Maurer, Ivona Brandic, and Rizos Sakellariou. "Self-Adaptive and Resource-Efficient SLA Enactment for Cloud Computing Infrastructures". In: *5th IEEE International Conference on Cloud Computing (CLOUD)*. IEEE, 2012, pp. 368–375. DOI: 10.1109/CLOUD.2012.55.

[190] Warren S. McCulloch and Walter Pitts. "A Logical Calculus of the Ideas Immanent in Nervous Activity". In: *The Bulletin of Mathematical Biophysics* 5.4 (1943), pp. 115–133. DOI: 10.1007/BF02478259.

[191] Peter M. Mell and Timothy Grance. *The NIST Definition of Cloud Computing*. Tech. rep. SP 800-145. National Institute of Standards & Technology, 2011.

[192] Daniel A. Menascé. “TPC-W: A Benchmark for E-Commerce”. In: *IEEE Internet Computing* 6.3 (2002), pp. 83–87. DOI: 10.1109/MIC.2002.1003136.

[193] Gabriele Mencagli, Marco Vanneschi, and Emanuele Vespa. “A Cooperative Predictive Control Approach to Improve the Reconfiguration Stability of Adaptive Distributed Parallel Applications”. In: *ACM Transactions on Autonomous and Adaptive Systems* 9.1 (2014), article 2. DOI: 10.1145/2567929.

[194] Xiaoqiao Meng, Vasileios Pappas, and Li Zhang. “Improving the Scalability of Data Center Networks with Traffic-Aware Virtual Machine Placement”. In: *29th IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2010, pp. 1–9. DOI: 10.1109/INFCOM.2010.5461930.

[195] Mark F. Mergen, Volkmar Uhlig, Orran Krieger, and Jimi Xenidis. “Virtualization for High-Performance Computing”. In: *ACM SIGOPS Operating Systems Review* 40.2 (2006), pp. 8–11. DOI: 10.1145/1131322.1131328.

[196] *Metronome: Owner’s Manual*. URL: https://www.metronome.io/pdf/owners_manual.pdf. White Paper. Version 0.967, 2018-04-17. Accessed 2019-05-05.

[197] Daniel Moldovan, Georgiana Copil, Hong-Linh Truong, and Schahram Dustdar. “MELA: Monitoring and Analyzing Elasticity of Cloud Services”. In: *5th IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE. 2013, pp. 80–87. DOI: 10.1109/CloudCom.2013.18.

[198] Justin D Moore, Jeffrey S Chase, Parthasarathy Ranganathan, and Ratnesh K Sharma. “Making Scheduling "Cool": Temperature-Aware Workload Placement in Data Centers”. In: *USENIX Annual Technical Conference (ATEC)*. USENIX, 2005, pp. 61–75.

[199] Laura R. Moore, Kathryn Bean, and Tariq Ellahi. “Transforming Reactive Auto-Scaling into Proactive Auto-Scaling”. In: *3rd International Workshop on Cloud Data and Platforms (CloudDP)*. ACM. 2013, pp. 7–12. DOI: 10.1145/2460756.2460758.

[200] Henry Muccini, Mohammad Sharaf, and Danny Weyns. “Self-Adaptation for Cyber-Physical Systems: A Systematic Literature Review”. In: *11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. ACM. 2016, pp. 75–81. DOI: 10.1145/2897053.2897069.

[201] Michael zur Muehlen and Robert Shapiro. "Business Process Analytics". In: *Handbook on Business Process Management 2*. Springer, 2015, pp. 243–263. DOI: 10.1007/978-3-642-45103-4_10.

[202] Gero Mühl, Ludger Fiege, and Peter Pietzuch. *Distributed Event-Based Systems*. Springer, 2006. DOI: 10.1007/3-540-32653-7.

[203] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. 2008. URL: https://bitcoin.org/bitcoin.pdf. White Paper. Accessed 2019-05-05.

[204] Christian Napoli, Giuseppe Pappalardo, and Emiliano Tramontana. "A Hybrid Neuro-Wavelet Predictor for QoS Control and Stability". In: *Congress of the Italian Association for Artificial Intelligence (AI*IA)*. LNCS 8249. Springer. 2013, pp. 527–538. DOI: 10.1007/978-3-319-03524-6_45.

[205] Yurii Nesterov. "A Method of Solving a Convex Programming Problem with Convergence Rate $O(1/k^2)$". In: *Soviet Mathematics Doklady* 27.2 (1983), pp. 372–376. ZBL: 0535.90071.

[206] Yurii Nesterov. "Introductory Lectures on Convex Optimization". In: *Applied Optimization* 87 (2004). ZBL: 1086.90045.

[207] Kalin Ovtcharov, Olatunji Ruwase, Joo-Young Kim, Jeremy Fowers, Karin Strauss, and Eric S. Chung. *Accelerating Deep Convolutional Neural Networks using Specialized Hardware*. 2015. URL: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/CNN20Whitepaper.pdf. White Paper, Microsoft Research. Version 2015-02-22. Accessed 2019-05-05.

[208] Claus Pahl and Brian Lee. "Containers and Clusters for Edge Cloud Architectures – A Technology Review". In: *3rd International Conference on Future Internet of Things and Cloud*. IEEE. 2015, pp. 379–386. DOI: 10.1109/FiCloud.2015.35.

[209] Jigar Patel, Sahil Shah, Priyank Thakkar, and K. Kotecha. "Predicting Stock and Stock Price Index Movement using Trend Deterministic Data Preparation and Machine Learning Techniques". In: *Expert Systems with Applications* 42.1 (2015), pp. 259–268. DOI: 10.1016/j.eswa.2014.07.040.

[210] Riccardo Petrolo, Valeria Loscri, and Nathalie Mitton. "Towards a Smart City Based on Cloud of Things, a Survey on the Smart City Vision and Paradigms". In: *Transactions on Emerging Telecommunications Technologies* 28.1 (2017), article e2931. DOI: 10.1002/ett.2931.

[211] Anastasiia Pika, Wil M. P. van der Aalst, Colin J. Fidge, Arthur H. M. ter Hofstede, and Moe T. Wynn. "Predicting Deadline Transgressions Using Event Logs". In: *International Conference on Business Process Management (BPM)*. LNBIP 132. Springer, 2013, pp. 211–216. DOI: 10.1007/978-3-642-36285-9_22.

[212] Anastasiia Pika, Wil M. P. van der Aalst, Moe Thandar Wynn, Colin J. Fidge, and Arthur H. M. ter Hofstede. "Evaluating and Predicting Overall Process Risk Using Event Logs". In: *Information Sciences* 352–353 (2016), pp. 98–120. DOI: 10.1016/j.ins.2016.03.003.

[213] David Martin Powers. "Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness and Correlation". In: *Journal of Machine Learning Technologies* 2.1 (2011), pp. 37–63. ISSN: 2229-3981.

[214] Kevin L. Priddy and Paul E. Keller. *Artificial Neural Networks: an Introduction*. Vol. 68. SPIE, 2005. ISBN: 978-0-819-45987-9.

[215] Michael O. Rabin. "Probabilistic Automata". In: *Information and Control* 6.3 (1963), pp. 230–245. DOI: 10.1016/S0019-9958(63)90290-0.

[216] Carl Edward Rasmussen. "Gaussian Processes in Machine Learning". In: *Summer School on Machine Learning (ML)*. LNCS 3176. Springer. 2003, pp. 63–71. DOI: 10.1007/978-3-540-28650-9_4.

[217] Siraj Raval. *Decentralized Applications: Harnessing Bitcoin's Blockchain Technology*. O'Reilly, 2016. ISBN: 978-1491924549.

[218] William John Reichmann. *Use and Abuse of Statistics*. Penguin, 1964. ISBN: 978-0-140-20707-1.

[219] Sebastian Rohjans, Christian Dänekas, and Mathias Uslar. "Requirements for Smart Grid ICT-Architectures". In: *3rd IEEE PES Innovative Smart Grid Technologies Europe (ISGT)*. IEEE, 2012, pp. 1–8. DOI: 10.1109/ISGTEurope.2012.6465617.

[220] Raúl Rojas. *Neural Networks: A Systematic Introduction*. Springer, 2013. DOI: 10.1007/978-3-642-61068-4.

[221] Anne Rozinat and Wil M. P. van der Aalst. "Decision Mining in ProM". In: *International Conference on Business Process Management (BPM)*. LNCS 4102. Springer, 2006, pp. 420–425. DOI: 10.1007/11841760_33.

[222] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning Representations by Back-Propagating Errors". In: *Cognitive Modeling*. MIT Press, 1986. ISBN: 9-780-2-626-6116-4.

[223] Mohammad Sadoghi, Martin Jergler, Hans-Arno Jacobsen, Richard Hull, and Roman Vaculín. "Safe Distribution and Parallel Execution of Data-

Centric Workflows over the Publish/Subscribe Abstraction". In: *IEEE Transactions on Knowledge and Data Engineering* 27.10 (2015), pp. 2824–2838. DOI: 10.1109/TKDE.2015.2421331.

[224] Felix Salfner and Miroslaw Malek. "Using Hidden Semi-Markov Models for Effective Online Failure Prediction". In: *26th IEEE International Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 2007, pp. 161–174. DOI: 10.1109/SRDS.2007.35.

[225] Arto Salomaa and Ian N. Sneddon. *Theory of Automata*. Pergamon, 1969. ISBN: 978-0-080-13376-8.

[226] Zohreh Sanaei, Saeid Abolfazli, Abdullah Gani, and Rajkumar Buyya. "Heterogeneity in Mobile Cloud Computing: Taxonomy and Open Challenges". In: *IEEE Communications Surveys & Tutorials* 16.1 (2014), pp. 369–392. DOI: 10.1109/SURV.2013.050113.00090.

[227] Madhulina Sarkar, Triparna Mondal, Sarbani Roy, and Nandini Mukherjee. "Resource Requirement Prediction using Clone Detection Technique". In: *Future Generation Computer Systems* 29.4 (2013), pp. 936–952. DOI: 10.1016/j.future.2012.09.010.

[228] Ramadass Sathya and Annamma Abraham. "Comparison of Supervised and Unsupervised Learning Algorithms for Pattern Classification". In: *International Journal of Advanced Research in Artificial Intelligence* 2.2 (2013), pp. 34–38. DOI: 10.14569/IJARAI.2013.020206.

[229] Stefan Schulte, Philipp Hoenisch, Christoph Hochreiner, Schahram Dustdar, Matthias Klusch, and Dieter Schuller. "Towards Process Support for Cloud Manufacturing". In: *18th IEEE International Enterprise Distributed Object Computing Conference (EDOC)*. IEEE, 2014, pp. 142–149. URL: 10.1109/EDOC.2014.28.

[230] Stefan Schulte, Philipp Hoenisch, Srikumar Venugopal, and Schahram Dustdar. "Introducing the Vienna Platform for Elastic Processes". In: *International Conference on Service-Oriented Computing (ICSOC)*. LNCS 7759. Springer, 2013, pp. 179–190. DOI: 10.1007/978-3-642-37804-1_19.

[231] Stefan Schulte, Philipp Hoenisch, Srikumar Venugopal, and Schahram Dustdar. "Realizing Elastic Processes with ViePEP". In: *International Conference on Service-Oriented Computing (ICSOC)*. LNCS 7759. Springer, 2013, pp. 439–442. DOI: 10.1007/978-3-642-37804-1_48.

[232] Stefan Schulte, Christian Janiesch, Srikumar Venugopal, Ingo Weber, and Philipp Hoenisch. "Elastic Business Process Management: State of the Art and Open Challenges for BPM in the Cloud". In: *Future Generation*

Computer Systems 46 (2015), pp. 36–50. DOI: 10.1016/j.future.2014.09.005.

[233] Bernd Schwegmann, Martin Matzner, and Christian Janiesch. "A Method and Tool for Predictive Event-Driven Process Analytics". In: *Wirtschaftsinformatik*. 2013, pp. 721–735.

[234] Fabrizio Sebastiani. "Machine Learning in Automated Text Categorization". In: *ACM Computing Surveys* 34.1 (2002), pp. 1–47. DOI: 10.1145/505282.505283.

[235] Ivan Selesnick. *Total Variation Denoising (an MM Algorithm)*. 2012. NYU Polytechnic School of Engineering Lecture Notes.

[236] Zhiming Shen, Sethuraman Subbiah, Xiaohui Gu, and John Wilkes. "CloudScale: Elastic Resource Scaling for Multi-Tenant Cloud Systems". In: *2nd ACM Symposium on Cloud Computing (SOCC)*. ACM. 2011, article 5. DOI: 10.1145/2038916.2038921.

[237] Marten Sigwart, Christoph Hochreiner, Michael Borkowski, and Stefan Schulte. *FakeLoad: An Open-Source Load Generator*. Tech. rep. TUV-1942-2018-01. Distributed Systems Group, TU Wien, 2018.

[238] Sukgpal Singh and Inderveer Chana. "QoS-Aware Autonomic Resource Management in Cloud Computing: A Systematic Review". In: *ACM Computing Surveys* 48.3 (2016), article 42. DOI: 10.1145/2843889.

[239] Olena Skarlat, Matteo Nardelli, Stefan Schulte, Michael Borkowski, and Philipp Leitner. "Optimized IoT Service Placement in the Fog". In: *Service Oriented Computing and Applications* 11.4 (2017), pp. 427–443. DOI: 10.1007/s11761-017-0219-8.

[240] Marina Sokolova, Nathalie Japkowicz, and Stan Szpakowicz. "Beyond Accuracy, F-Score and ROC: A Family of Discriminant Measures for Performance Evaluation". In: *Australian Joint Conference on Artificial Intelligence (AI)*. LNCS 4304. Springer, 2006, pp. 1015–1021. DOI: 10.1007/11941439_114.

[241] Jorge Sola and José Sevilla. "Importance of Input Data Normalization for the Application of Neural Networks to Complex Industrial Problems". In: *IEEE Transactions on Nuclear Science* 44.3/3 (1997), pp. 1464–1468. DOI: 10.1109/23.589532.

[242] Florian Stertz, Stefanie Rinderle-Ma, Tobias Hildebrandt, and Jürgen Mangler. "Testing Processes with Service Invocation: Advanced Logging in CPEE". In: *International Conference on Service-Oriented Computing (ICSOC)*. LNCS 10380. Springer, 2016, pp. 189–193. DOI: 10.1007/978-3-319-68136-8_22.

[243] Gerhard Stürmer, Jürgen Mangler, and Erich Schikuta. "Building a Modular Service Oriented Workflow Engine". In: *IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*. IEEE, 2009, pp. 1–4. DOI: 10.1109/SOCA.2009.5410270.

[244] Suriadi Suriadi, Chun Ouyang, Wil M. P. van der Aalst, and Arthur H. M. ter Hofstede. "Root Cause Analysis with Enriched Process Logs". In: *International Conference on Business Process Management (BPM)*. LNBIP 132. Springer, 2013, pp. 174–186. DOI: 10.1007/978-3-642-36285-9_18.

[245] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. "On the Importance of Initialization and Momentum in Deep Learning". In: *International Conference on Machine Learning (ICML)*. PMLR, 2013, pp. 1139–1147. ISBN: 978-1-629-93306-1.

[246] Melanie Swan. "Blockchain for Business: Next-Generation Enterprise Artificial Intelligence Systems". In: *Advances in Computers* 111 (2018), pp. 121–162. DOI: 10.1016/bs.adcom.2018.03.013.

[247] Claudia Szabo and Trent Kroeger. "Evolving Multi-objective Strategies for Task Allocation of Scientific Workflows on Public Clouds". In: *IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2012, pp. 1–8. DOI: 10.1109/CEC.2012.6256556.

[248] Andrew S. Tanenbaum and Maarten van Steen. *Distributed Systems: Principles and Paradigms*. Prentice-Hall, 2007. ISBN: 978-1-530-28175-6.

[249] Feilong Tang, Minyi Guo, Mianxiong Dong, Minglu Li, and Hu Guan. "Towards Context-Aware Workflow Management for Ubiquitous Computing". In: *International Conference on Embedded Software and Systems (ICESS)*. IEEE, 2008, pp. 221–228. DOI: 10.1109/ICESS.2008.83.

[250] Irene Teinemaa, Marlon Dumas, Fabrizio Maria Maggi, and Chiara Di Francescomarino. "Predictive Business Process Monitoring with Structured and Unstructured Data". In: *International Conference on Business Process Management (BPM)*. LNCS 9850. Springer, 2016, pp. 401–417. DOI: 10.1007/978-3-319-45348-4_23.

[251] Dennis Upper. "The Unsuccessful Self-treatment of a Case of Writer's Block". In: *Journal of Applied Behavior Analysis* 7.3 (1974), pp. 497–497. DOI: 10.1901/jaba.1974.7-497a.

[252] Haleh Vafaie and Kenneth De Jong. "Genetic Algorithms as a Tool for Feature Selection in Machine Learning". In: *4th International Conference on Tools with Artificial Intelligence (TAI)*. 1992, pp. 200–203. DOI: 10.1109/TAI.1992.246402.

[253] Mohammad Valipour, Mohammad Ebrahim Banihabib, and Seyyed Mahmood Reza Behbahani. "Comparison of the ARMA, ARIMA, and the Autoregressive Artificial Neural Network Models in Forecasting the Monthly Inflow of Dez Dam Reservoir". In: *Journal of Hydrology* 476 (2013), pp. 433–441. DOI: 10.1016/j.jhydrol.2012.11.017.

[254] Luis M. Vaquero and Luis Rodero-Merino. "Finding your Way in the Fog: Towards a Comprehensive Definition of Fog Computing". In: *ACM SIGCOMM Computer Communication Review* 44.5 (2014), pp. 27–32. DOI: 10.1145/2677046.2677052.

[255] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. "A Break in the Clouds: Towards a Cloud Definition". In: *ACM SIGCOMM Computer Communication Review* 39.1 (2008), pp. 50–55. DOI: 10.1145/1496091.1496100.

[256] Nedeljko Vasić, Dejan Novaković, Svetozar Miučin, Dejan Kostić, and Ricardo Bianchini. "DejaVu: Accelerating Resource Allocation in Virtualized Environments". In: *17th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM. 2012, pp. 423–436. DOI: 10.1145/2150976.2151021.

[257] Chemuduri Viswanath and C. Valliyammai. "CPU Load Prediction using ANFIS for Grid Computing". In: *IEEE International Conference On Advances In Engineering, Science And Management (ICAESM)*. IEEE. 2012, pp. 343–348. ISBN: 978-81-909042-2-3.

[258] Marko Vukolić. "The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication". In: *International Workshop on Open Problems in Network Security*. LNCS 9591. Springer, 2015, pp. 112–125. DOI: 10.1007/978-3-319-39028-4_9.

[259] Lizhe Wang, Gregor Von Laszewski, Jai Dayal, Xi He, Andrew J Younge, and Thomas R Furlani. "Towards Thermal Aware Workload Scheduling in a Data Center". In: *10th International Symposium on Pervasive Systems, Algorithms, and Networks (ISPAN)*. IEEE. 2009, pp. 116–122. DOI: 10.1109/I-SPAN.2009.22.

[260] Will Warren and Amir Bandeali. *0x: An Open Protocol for Decentralized Exchange on the Ethereum Blockchain*. URL: https://0xproject.com/pdfs/0x_white_paper.pdf. White Paper. Version 2017-02-21. Accessed 2019-05-05.

[261] Christopher J. C. H. Watkins and Peter Dayan. "Q-Learning". In: *Machine Learning* 8.3-4 (1992), pp. 279–292. DOI: 10.1007/BF00992698.

[262] Bernard Lewis Welch. "The Generalization of 'Student's' Problem when Several Different Population Variances are Involved". In: *Biometrika* 34.1-2 (1947), pp. 28–35. DOI: 10.1093/biomet/34.1-2.28.

[263] Mathias Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer, 2012. DOI: 10.1007/978-3-540-73522-9.

[264] Gerhard Widmer and Miroslav Kubat. "Learning in the Presence of Concept Drift and Hidden Contexts". In: *Machine Learning* 23.1 (1996), pp. 69–101. DOI: 10.1023/A:1018046501280.

[265] Roel Wieringa. *Design Science Methodology for Information Systems and Software Engineering*. Springer, 2014. DOI: 10.1007/978-3-662-43839-8.

[266] J. R. Willett, Maran Hidskes, David Johnston, Ron Gross, and Marv Schneider. *Omni Protocol Specification*. 2017. URL: https://github.com/OmniLayer/spec. Version 0.5, 2017-01-23. Accessed 2019-05-05.

[267] Gavin Wood. *PolkaDot: Vision for a Heterogeneous Multi-Chain Framework*. 2016. URL: https://polkadot.network/PolkaDotPaper.pdf. White Paper. Draft 1, Version 0.1.0, 2016-11-10. Accessed 2019-05-05.

[268] Linlin Wu, Saurabh Kumar Garg, and Rajkumar Buyya. "SLA-Based Resource Allocation for Software as a Service Provider (SaaS) in Cloud Computing Environments". In: *11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. IEEE, 2011, pp. 195–204. DOI: 10.1109/CCGrid.2011.51.

[269] Jielong Xu, Zhenhua Chen, Jian Tang, and Sen Su. "T-Storm: Traffic-Aware Online Scheduling in Storm". In: *34th IEEE International Conference on Distributed Computing Systems (ICDCS)*. IEEE. 2014, pp. 535–544. DOI: 10.1109/ICDCS.2014.61.

[270] Hongming Yang, Jun Yi, Junhua Zhao, and ZhaoYang Dong. "Extreme Learning Machine Based Genetic Algorithm and its Application in Power System Economic Dispatch". In: *Neurocomputing* 102 (2013), pp. 154–162. DOI: 10.1016/j.neucom.2011.12.054.

[271] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzales, Scott Shenker, and Ion Stoica. "Apache Spark: A Unified Engine for Big Data Processing". In: *Communications of the ACM* 59.11 (2016), pp. 56–65. DOI: 10.1145/2934664.

[272] Ahmed I Zayed. *Handbook of Function and Generalized Function Transformations*. CRC, 1996. ISBN: 978-0-849-38076-1.

[273] Zhi-Hui Zhan, Xiao-Fang Liu, Yue-Jiao Gong, Jun Zhang, Henry Shu-Hung Chung, and Yun Li. "Cloud Computing Resource Scheduling and a Survey of Its Evolutionary Approaches". In: *ACM Computing Surveys (CSUR)* 47.4 (2015), article 63. DOI: 10.1145/2788397.

[274] Qi Zhang, Lu Cheng, and Raouf Boutaba. "Cloud Computing: State-of-the-Art and Research Challenges". In: *Journal of Internet Services and Applications* 1.1 (2010), pp. 7–18. DOI: 10.1007/s13174-010-0007-6.

[275] Taiyang Zhang and Loong Wang. *Republic Protocol: A decentralized dark pool exchange providing atomic swaps for Ethereum-based assets and Bitcoin.* URL: https://releases.republicprotocol.com/whitepaper/1.0.0/whitepaper_1.0.0.pdf. White Paper. Version 2017-12-18. Accessed 2019-05-05.

[276] Zhili Zhao and Adrian Paschke. "Event-Driven Scientific Workflow Execution". In: *International Conference on Business Process Management (BPM)*. LNBIP 132. Springer, 2012, pp. 390–401. DOI: 10.1007/978-3-642-36285-9_42.

[277] Zibin Zheng and Michael R. Lyu. "Selecting an Optimal Fault Tolerance Strategy for Reliable Service-Oriented Systems with Local and Global Constraints". In: *IEEE Transactions on Computers* 64 (2015), pp. 219–232. DOI: 10.1109/TC.2013.189.

[278] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Xiangping Chen, and Huaimin Wang. "Blockchain Challenges and Opportunities: A Survey". In: *International Journal of Web and Grid Services* 14.4 (2018), pp. 352–375. DOI: 10.1504/IJWGS.2018.095647.

[279] Zibin Zheng, Shaoan Xie, Hongning Dai, Xiangping Chen, and Huaimin Wang. "An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends". In: *IEEE International Congress on Big Data*. IEEE. 2017, pp. 557–564. DOI: 10.1109/BigDataCongress.2017.85.

[280] Aviv Zohar. "Bitcoin: Under the Hood". In: *Communications of the ACM* 58.9 (2015), pp. 104–113. DOI: 10.1145/2701411.

APPENDIX A

Curriculum Vitæ

Address:	Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR) Lilienthalplatz 7 38108 Braunschweig Germany
Website:	`www.borkowski.at`
E-Mail:	`michael@borkowski.at`

Education

2015/11 – *ongoing*	Computer Science (Doctoral Studies)
2012/09 – 2015/10	Software Engineering & Internet Computing (Master Studies)
2009/10 – 2012/09	Software and Information Engineering (Bachelor Studies)

Professional Experience

2019/05 – *ongoing*	Research Scientist, German Aerospace Center (DLR)
2015/11 – 2019/04	Research Assistant, TU Wien
2015/05 – 2015/10	Software Developer, EclipseSource Services GmbH
2013/12 – 2015/04	Systems Architect & DevOps, Flatout Technologies GmbH
2010/10 – 2014/02	Teaching Assistant, TU Wien
2010/08 – 2010/09	Internship, ASFINAG Service GmbH
2008/07	Internship, Siemens AG Österreich
2007/07	Internship, Siemens AG Österreich

Research

Fields of Interest: Optimization of cost, performance and resource utilization in cloud computing; cloud manufacturing, Industry 4.0, Industrial IoT; prediction-based proactive systems; machine learning; blockchain technologies.

Projects

2019/05 – *ongoing*	City-ATM (DLR Institutional Funding) Demonstration of Traffic Management in Urban Airspace
2019/05 – *ongoing*	RESPONDRONE (EU H2020 RIA) Novel Integrated Solution of Operating a Fleet of Drones with Multiple Synchronized Missions for Disaster Responses
2018/03 – 2019/04	TAST (Bitpanda GmbH/Pantos GmbH) Token Atomic Swap Technology
2015/11 – 2017/12	CREMA (EU H2020 RIA) Cloud-based Rapid Elastic Manufacturing
2014/03 – 2014/11	SIMPLI-CITY (EU FP7) The Road User Information System of the Future

Scientific Activities and Services

- ZEUS Workshop 2019 and 2020: Member of the Program Committee
- Elsevier Robotics and Computer-Integrated Manufacturing: Reviewer
- ACM Transactions on the Web: Reviewer
- IEEE Transactions on Cloud Computing: Reviewer
- IEEE Transactions on Services Computing: Reviewer
- IEEE Communications Surveys and Tutorials: Reviewer
- Wiley Concurrency and Computation: Practice and Experience: Reviewer

Assistance Supervision of Theses

- Dragan Tosic: Blockchain Interoperability: A Cross-Chain Token Transfer Protocol. Diploma Thesis, TU Wien (ongoing).
- Sabine Weninger: Data Prefetching in Smart Systems, Bachelor's Thesis. TU Wien, April 2018.
- Christian Schubert: Trustworthy Measurement and Arbitration of Service Level Agreements in the Cloud. Diploma Thesis, TU Wien, January 2018.

Publications

The following list shows the author's publications to date. An updated list of publications is available on the author's website[1].

Journal Articles

- Michael Borkowski, Marten Sigwart, Philipp Frauenthaler, Taneli Hukkinen, and Stefan Schulte. "DeXTT: Deterministic Cross-Blockchain Token Transfers". In: *IEEE Access* 7.1 (2019), pp. 111030–111042. DOI: 10.1109/ACCESS.2019.2934707.
- Michael Borkowski, Christoph Hochreiner, and Stefan Schulte. "Minimizing Cost by Reducing Scaling Operations in Distributed Stream Processing". In: *PVLDB* 12.7 (2019), pp. 724–737. DOI: 10.14778/3317315.3317316.
- Michael Borkowski, Walid Fdhila, Matteo Nardelli, Stefanie Rinderle-Ma, and Stefan Schulte. "Event-Based Failure Prediction in Distributed Business Processes". In: *Information Systems* 81 (2019), pp. 220–235. DOI: 10.1016/j.is.2017.12.005.
- Olena Skarlat, Matteo Nardelli, Stefan Schulte, Michael Borkowski, and Philipp Leitner. "Optimized IoT Service Placement in the Fog". In: *Service Oriented Computing and Applications* 11.4 (2017), pp. 427–443. DOI: 10.1007/s11761-017-0219-8.

Conference and Workshop Proceedings

- Philipp Frauenthaler, Michael Borkowski, and Stefan Schulte. "A Framework for Assessing and Selecting Blockchains at Runtime (short paper, accepted for publication)". In: *IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*. 2020, nn–nn.
- Marten Sigwart, Michael Borkowski, Marco Peise, Stefan Schulte, and Stefan Tai. "Blockchain-Based Data Provenance for the Internet of Things". In: *9th International Conference on the Internet of Things (IoT)*. 2019, pp. 1–8. DOI: 10.1145/3365871.3365886.
- Vasileios Karagiannis, Alexandre Venito, Rodrigo Coelho, Michael Borkowski, and Gerhard Fohler. "Edge Computing with Peer to Peer Interactions: Use Cases and Impact". In: *Workshop on Fog Computing and the Internet of Things (Fog-IoT)*. 2019, pp. 1–5. DOI: 10.1145/3313150.3313226.

[1] http://www.borkowski.at/

- Sabine Weninger and Michael Borkowski. "Data Prefetching in Smart Systems". In: *22nd IEEE International Enterprise Distributed Object Computing Conference (EDOC)*. 2018, pp. 204–207. DOI: 10.1109/EDOCW.2018.00037. Demo paper.
- Christian Schubert, Michael Borkowski, and Stefan Schulte. "Trustworthy Detection and Arbitration of SLA Violations in the Cloud". In: *7th European Conference on Service-Oriented and Cloud Computing (ESOCC)*. LNCS 11116. 2018, pp. 5–16. DOI: 10.1007/978-3-319-99819-0_7.
- Philipp Waibel, Svetoslav Videnov, Michael Borkowski, Christoph Hochreiner, Stefan Schulte, and Jan Mendling. "Process Simulation for Machine Reservation in Cloud Manufacturing". In: *16th IEEE International Conference on Industrial Informatics (INDIN)*. 2018, pp. 270–277. DOI: 10.1109/INDIN.2018.8472038.
- Michael Borkowski, Christoph Hochreiner, and Stefan Schulte. "Moderated Resource Elasticity for Stream Processing Applications". In: *Parallel Processing Workshops (Euro-Par)*. LNCS 10659. Springer, 2017, pp. 5–16. DOI: 10.1007/978-3-319-75178-8_1.
- Michael Borkowski, Stefan Schulte, and Christoph Hochreiner. "Predicting Cloud Resource Utilization". In: *9th IEEE/ACM International Conference on Utility and Cloud Computing (UCC)*. IEEE/ACM, 2016, pp. 37–42. DOI: 10.1145/2996890.2996907.
- Olena Skarlat, Stefan Schulte, Michael Borkowski, and Philipp Leitner. "Resource Provisioning for IoT Services in the Fog". In: *9th IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*. 2016, pp. 32–39. DOI: 10.1109/SOCA.2016.10.
- Michael Borkowski, Olena Skarlat, Stefan Schulte, and Schahram Dustdar. "Prediction-Based Prefetch Scheduling in Mobile Service Applications". In: *2016 IEEE International Conference on Mobile Services (MS)*. 2016, pp. 41–48. DOI: 10.1109/MobServ.2016.17.
- Christoph Hochreiner, Philipp Waibel, and Michael Borkowski. "Bridging gaps in cloud manufacturing with 3D printing". In: *Informatik 2016, 46. Jahrestagung der Gesellschaft für Informatik*. Lecture Notes in Informatics vol. 259. 2016, pp. 1623–1626.
- Olena Skarlat, Michael Borkowski, and Stefan Schulte. "Towards a methodology and instrumentation toolset for cloud manufacturing". In: *1st International Workshop on Cyber-Physical Production Systems (CPPS)*. IEEE. 2016, pp. 1–4.
- Stefan Schulte, Michael Borkowski, Christoph Hochreiner, Matthias Klusch, Aitor Murguzur, Olena Skarlat, and Philipp Waibel. "Bringing Cloud-based Rapid Elastic Manufacturing to Reality with CREMA". in: *Workshop on Intelligent Systems Configuration Services for Flexible Dynamic Global Production Networks (FLEXINET)*. Springer. 2016, pp. 407–413.

Unrefereed Papers

- Stefan Schulte, Marten Sigwart, Philipp Frauenthaler, and Michael Borkowski. "Towards Blockchain Interoperability". In: *BPM Blockchain and Central and Eastern Europe Forum*. LNBIP 361. Springer. 2019, pp. 3–10. DOI: 10.1007/978-3-030-30429-4_1.
- Michael Borkowski, Marten Sigwart, Philipp Frauenthaler, Taneli Hukkinen, and Stefan Schulte. *DeXTT: Deterministic Cross-Blockchain Token Transfers*. 2019. arXiv, Article 1905.06204.
- Michael Borkowski, Philipp Frauenthaler, Marten Sigwart, Taneli Hukkinen, Oskar Hladký, and Stefan Schulte. *Cross-Blockchain Technologies: Review, State of the Art, and Outlook*. 2019. DOI: 10.13140/RG.2.2.30902.14403. White Paper, TU Wien.
- Michael Borkowski, Christoph Ritzer, and Stefan Schulte. *Deterministic Witnesses for Claim-First Transactions*. 2018. DOI: 10.13140/RG.2.2.17480.37123. White Paper, Technische Universität Wien.
- Marten Sigwart, Christoph Hochreiner, Michael Borkowski, and Stefan Schulte. *FakeLoad: An Open-Source Load Generator*. Tech. rep. TUV-1942-2018-01. Distributed Systems Group, TU Wien, 2018.
- Michael Borkowski, Christoph Ritzer, Daniel McDonald, and Stefan Schulte. *Caught in Chains: Claim-First Transactions for Cross-Blockchain Asset Transfers*. 2018. DOI: 10.13140/RG.2.2.24191.25769. White Paper, TU Wien.
- Michael Borkowski, Daniel McDonald, Christoph Ritzer, and Stefan Schulte. *Towards Atomic Cross-Chain Token Transfers: State of the Art and Open Questions within TAST*. 2018. DOI: 10.13140/RG.2.2.10769.48489. White Paper, TU Wien.
- Michael Borkowski, Walid Fdhila, Matteo Nardelli, Stefanie Rinderle-Ma, and Stefan Schulte. "Event-based Failure Prediction in Distributed Business Processes". In: *CoRR* arXiv:1712.08342 (2018).

Theses

- "Smart Prefetching for Mobile Users under Volatile Network Conditions". Diploma Thesis. TU Wien, 2015
- "ACTA in a Nutshell: Das Handelsabkommen ACTA in seinen wichtigsten Zügen". Bachelor's Thesis. TU Wien, 2012