Institute of Software Engineering
Software Quality and Architecture

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Architecture-based Simulation of Elasticity Policies with Learning Capabilities

Jens Berberich

Course of Study:	Informatik
Examiner:	Prof. Dr.-Ing. Steffen Becker
Supervisor:	Floriment Klinaku, M.Sc., Sarah Sophie Stieß, M.Sc.
Commenced:	June 11, 2024
Completed:	December 11, 2024

Abstract

Context. For cloud-native applications, the choice of scaling method can highly impact both the application performance and operational cloud costs. To be able to evaluate different cloud application architectures at design time, an architecture can be modeled and simulated. Palladio [BKR07] is a modeling language and simulator that allows the modeling of architectures for performance predictions at design time. The Slingshot approach [KKSB23] proposes adding a new type of architectural view called Scaling Policy Definitions (SPDs) to Palladio for model-based performance predictions of elasticity policies. In addition, the Slingshot approach encompasses the Slingshot simulator which employs discrete event simulations of fixed length to determine the quality impact of modeled policies.
Problem. Resource provisioning and autoscaling in the cloud may rely on advanced scaling approaches that are not only reactive but also include predictive components that are often learning-based. However, the Slingshot approach currently supports only the modeling of elasticity policies that can be classified as reactive.
Objective. The objective of this thesis is to identify currently used learning-based elasticity policies and to implement them in Slingshot. Thus, this thesis extends the Slingshot approach with support for learning-based approaches and compares them both to each other and their Slingshot implementation with results provided in the original paper.
Method. The approaches are identified using a rapid literature review. For the extension of SPDs, a model-driven software development approach is followed. The evaluation is based on the evaluations done by the identified papers and is performed using Slingshot.
Result. The SPD metamodel is extended with support for learning-based elasticity policies and the Fuzzy SARSA and Fuzzy Q-Learning approaches are implemented in the Slingshot simulator. The evaluation shows that these policies perform better than the existing reactive scaling policies.
Conclusion. By implementing state-of-the-art predictive elasticity policies in Slingshot and extending SPDs with support for modeling them, this thesis makes Slingshot and the SPD language more complete. The evaluation shows that learning-based elasticity policies can offer improved adherence to Service Level Objectives (SLOs) while using fewer resources, but initially perform worse than reactive policies during the learning phase.

Kurzfassung

Kontext. Die Performance und die operationalen Kosten von Cloud-nativen Anwendungen hängen stark von dem verwendeten Skalierungsalgorithmus ab. Um verschiedene Architekturen von Cloud-Anwendungen zu vergleichen, können diese modelliert werden. Zur Modellierung und Simulation von Softwarearchitekturen kann Palladio [BKR07] eingesetzt werden. Palladio wird durch den Slingshot-Ansatz [KKSB23] um Scaling Policy Definitions (SPDs) erweitert, wodurch Performancevorhersagen auch für Modelle mit Skalierungsalgorithmen getroffen werden können. Der Slingshot-Ansatz umfasst auch den Slingshot-Simulator, der unter Verwendung von ereignisbasierten Simulationen die Auswirkungen der modellierten Regeln beurteilen kann.
Problem. Fortgeschrittene Algorithmen zur Ressourcenbereitstellung und automatischen Skalierung in der Cloud sind oft nicht rein reaktiv, sondern enthalten häufig auch prädiktive Komponenten, die oft selbstlernend sind. Der Slingshot-Ansatz unterstützt derzeit jedoch nur die Modellierung reaktiver Skalierungsregeln.
Zielsetzung. Ziel der Arbeit ist es, aktuell verwendete lernbasierte Skalierungsmethoden zu identifizieren und in Slingshot zu implementieren. Dazu wird der Slingshot-Ansatz so erweitert, dass lernbasierte Methoden unterstützt werden. Die in Slingshot implementierten Methoden werden dann untereinander, mit reaktiven Methoden und mit den Ergebnissen der Orginalmethode verglichen.
Methodik. Verschiedene Methoden werden durch einer Literaturauswertung identifiziert und verglichen. Die SPDs werden dann mit Hilfe eines modellgetriebenen Softwareentwicklungsansatzes erweitert. Für die Evaluation werden die Evaluationen der identifizierten Artikel in Slingshot nachgebildet.
Ergebnis. Das SPD Metamodell unterstützt nun lernbasierte Skalierungsmethoden. Die Methoden Fuzzy SARSA und Fuzzy Q-Learning wurden im Simulator Slingshot implementiert. Die Evaluierung zeigt, dass die prädiktiven Methoden eine Verbesserung gegenüber den existierenden reaktiven Skalierungsmethoden darstellen.
Zusammenfassung. Durch die Implementierung moderner prädiktiver Skalierungsmethoden in Slingshot und die Erweiterung von SPDs mit Modellierungsmethoden für diese neuen Skalierungsmethoden macht diese Arbeit Slingshot und das SPD-Modell vollständiger. Die Evaluierung zeigt, dass lernbasierte Skalierungsmethoden im Vergleich zu reaktiven Skalierungsmethoden besser in der Lage sind, Service Level Objectives (SLOs) zu erfüllen und dabei weniger Ressourcen benötigen, jedoch zunächst eine Lernphase mit schlechterem Verhalten erfordern.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

EMF Eclipse Modeling Framework. 4

EPC Event Process Chain, see [Van99]. ix

PCM Palladio Component Model. 1

QVTo QVT Operational. 4

RL Reinforcement Learning. 4

SLA Service Level Agreement. 19

SLO Service Level Objective. iii

SPD Scaling Policy Definition. iii

VM Virtual Machine. 3

1 Introduction

Cloud computing has become very popular, especially in the last two decades, so it is important to have tools that help guide decisions about the architecture of cloud-native applications. These tools can help by enabling the architecture of an application to be modeled (e.g., using the Palladio Component Model (PCM) [Bec10; BKR07]) so that performance predictions can be made using a simulator such as Slingshot. One of the main benefits of cloud computing is the ease of scalability, which is especially important for modern architectural styles such as microservices. For this, the Scaling Policy Definitions (SPDs) metamodel extends the PCM and supports the definition of scaling policies. The SPD models can be simulated using the Slingshot simulator [KKSB23].

The choice of scaling method can be highly impactful on both the performance of an application and on operational cloud costs. Cloud providers support not only reactive but also predictive scaling methods that try to anticipate the resource demand of a system [Ama; Mic24]. These predictive scaling methods are also discussed in thecurrent academic literature on scaling approaches [LML14, p. 563; GMP+21]. However, the Slingshot approach currently only supports reactive scaling approaches, which limits the applicability of SPDs and the Slingshot simulator. Additionally, there are not many performance-focused comparisons between the different scaling approaches.

This thesis thus extends the Slingshot simulator and the SPD metamodel to incorporate predictive, learning-based scaling methods. The implemented approaches are then compared with each other, with reactive scaling policies, and with the results from their original studies to gain a better understanding of the trade-offs associated with different predictive scaling strategies. They are additionally compared to the existing reactive scaling policies.

To find state-of-the-art predictive scaling methods, a rapid literature review was performed and the approaches were classified. Criteria for determining which approaches should be implemented in the Slingshot simulator are proposed and applied to find suitable models. Then, the SPD metamodel is extended to support the new predictive approaches. The approaches are implemented in the Slingshot simulator. Finally, the implemented models are evaluated using the Slingshot simulator and scenarios that are comparable to those that are performed in the papers previously identified in the literature review.

The contribution of this thesis is both the implementation of models enabling predictive elasticity policies in the Slingshot simulator and the extension of the SPD metamodel for this implementation as well as an evaluation. The model and implementation artifacts enable SPDs and Slingshot to cover new use cases. The evaluation assesses the validity of the approaches and compares their performance in Slingshot to the claims made in the corresponding papers and to reactive scaling policies. It shows that the model-based approaches can improve the scaling behavior compared to reactive scaling policies and reach a higher SLO adherence while using fewer resources. This comes with the drawback of first needing a training phase in which more scaling actions are performed, potentially leading to more SLO violations and increased resource usage in this initial phase.

The detailed research questions that are answered by this thesis are:

RQ1.1 Which of the state-of-the-art learning-based elasticity policies should be selected for modeling and simulation?

RQ2.1 How and at what abstraction level can the selected approaches be effectively implemented in Slingshot?

RQ3.1 How do the different approaches compare to each other when using the Slingshot simulator?

RQ3.2 Can the claims made in the accompanying papers be verified using the Slingshot simulator?

Thesis Structure

The thesis is structured into the following chapters:

Chapter 2 – Foundations: An overview of the foundations of the thesis topic including fundamentals used for the implemented approaches is provided.

Chapter 3 – Related Work: This chapter gives an overview of some significant related works and discusses why the problem of autoscaling for cloud applications has been insufficiently solved in the context of software architecture modeling frameworks.

Chapter 4 – Literature Review: The methodology and the results of the performed literature review are discussed. This chapter also gives the reasoning for the selected approaches.

Chapter 5 – Concept: The conceptual foundations for changing the SPD metamodel and extending the Slingshot simulator are given, including an overview of the chosen predictive models.

Chapter 6 – Implementation: Here, further details on the model extension and implementation are provided.

Chapter 7 – Evaluation: This chapter shows the methodology and results of the evaluation and discusses the results.

Chapter 8 – Conclusion The thesis is concluded by explaining the applicability of the results. Some ideas for further research are provided and possible threats to validity are discussed.

2 Foundations

The Foundations section describes the concept of modeling component-based architectures and highlights some significant work in the area of simulating these models. Additionally, Reinforcement Learning and Fuzzy Logic Systems are explained, as they are used in the implemented models.

2.1 Modeling Component-Based Architectures

The thesis is based on the topic of model-driven quality-of-service predictions for component-based architectures. Component-based software engineering provides support for component-based development, i.e. development of software that is composed of separate parts that can be deployed independently, have some specified interfaces and that will be used in a larger software system [Szy99, p. 34].

These component-based architectures can then be modeled using the Palladio Component Model (PCM) [Bec10; BKR07]. A PCM instance consists of several parts:

- The repository contains component specifications, i.e. all components, their Interfaces, and relationships between components.
- The system or assembly model specifies the instances of the components and how they are connected to each other and to the interfaces required or provided by the system as a whole.
- The resource environment models the resources on which an application will be deployed. It also models the performance characteristics such as latency, processing rate, or the used scheduling strategy for the specified resources.
- The allocation model maps the component instances modeled in the assembly model to resources that are specified in the resource environment model.
- The usage model contains usage scenarios that contain workloads modeling the behavior of users, such as the method calls made by a user, the number of users, their arrival time, or the total population.

Slingshot then builds on this model by adding Scaling Policy Definitions (SPDs) that describe how scaling should happen for a given PCM instance. Thus, SPDs enable building elasticity policies for PCMs such that cloud-native applications that rely on elasticity or scalability can be modeled [KKSB23]. Inside an SPD, a scaling policy that consists of a trigger, an adjustment action, and optional constraints describes the scaling behavior. SPDs support horizontal scaling, i.e., scaling by increasing or decreasing the count of replicas such as Virtual Machines (VMs) or containers.

With these models, a prediction of the performance of an architecture for the modeled workload can be made such that different architectures can be compared to each other and the best architecture can be chosen. When using the Slingshot approach and SPDs, the event-based Slingshot simulator will be used to simulate an architecture. It uses QVT Operational (QVTo) model transformations to perform the scaling actions.

For the development of SPD, the Eclipse Modeling Framework (EMF) is used [Ste09]. The EMF allows for model-driven development by modeling software conforming to the Ecore metamodel. This model can then be used by EMF.Codegen to build a complete editor for a given EMF model where the model components can then be implemented. In the Slingshot approach, the SPD is defined as an Ecore model with additional information being provided for the code generation in a genmodel.

2.2 Reinforcement Learning

Reinforcement Learning (RL) [SB20] is an unsupervised learning approach where an agent learns a behavior policy based on taking actions in an environment and observing rewards. The structure of an RL problem is shown in Figure 2.1. The agent gets a state S_i and reward R_i from the environment and then performs an action a_i. After the action has been performed, a new state S_{i+1} and a new reward R_{i+1} that are dependent on the performed action and possibly other environmental factors are observed by the agent.

To learn the policy, the agent keeps track of action or state values that map each state-action pair or state to a value. The action values are denoted as $q(s, a)$ and should estimate the expected return when in a state and performing an action. The policy then uses them to select an action based on the currently observed state. The epsilon-greedy policy is often used. It specifies that with a probability of ε, a random action is chosen. Otherwise, the action that corresponds to the maximum action value for the currently observed state is chosen. Choosing a random action is considered to be exploratory, as it can lead to the agent reaching different states that are not currently deemed optimal by the current values. Choosing the action with the highest associated action value means exploiting the learned knowledge that is encoded in the action values.

There can be different techniques for initializing the action values. For *pessimistic* and *optimistic* initialization, all action values are initially set to the same value, which is either lower or higher than the expected best action value. An optimistic initialization further encourages early exploration, as the action values decrease over time [SB20, p. 34]. The action values can be calculated incrementally using temporal difference learning. Q-Learning and SARSA are often used for this purpose. For Q-learning, the action values are updated using the following equation:

$$Q(S_t, a_t) \leftarrow Q(S_t, a_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, a_t) \right] \tag{2.1}$$

This strategy is considered to be off-policy because the update function does not depend on the used policy but instead updates the action values based on the maximal possible action value achievable in the next state.

A commonly used on-policy algorithm is SARSA:

$$Q(S_t, a_t) \leftarrow Q(S_t, a_t) + \alpha \left[R_{t+1} + \gamma Q(S_{t+1}, a_{t+1}) - Q(S_t, a_t) \right] \tag{2.2}$$

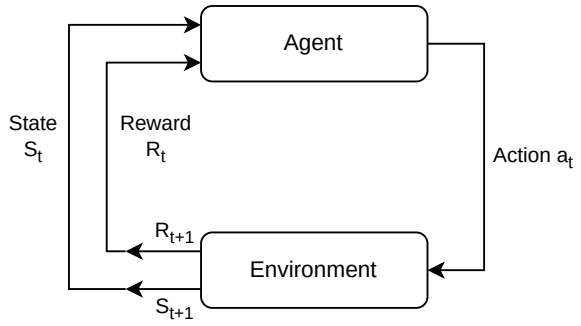


Figure 2.1: Representation of a Reinforcement Learning agent

Here, the update function considers the action a_{t+1} that is taken next in order to update the action values.

2.3 Fuzzy Logic Systems

Fuzzy logic systems generally are mappings of numerical input features into a numerical output value [Men95]. By using fuzzy sets where an input can belong to different sets in varying degrees, they are able to handle uncertain or imprecise environments. In this thesis, a fuzzy logic system is used to map a system state to a scaling action. A fuzzy logic system consists of three components: A *fuzzifier*, an *interference system*, and a *defuzzifier*.

The *fuzzifier* converts crisp input variables into fuzzy input sets, which in our case consist of linguistic variables. To get from a crisp input variable such as the workload specified as CPU utilization in % to a linguistic variable that is fuzzy, fuzzy sets and membership functions w are needed. The fuzzy sets consist of labels such as “high” or “medium” and a corresponding membership function. The membership function then determines the degree of membership for a given crisp input variable and a fuzzy label. For example, $\mu_{\text{“high”}}(0.8) = 0.6$ and $\mu_{\text{“medium”}}(0.8) = 0.4$, meaning that a utilization of 80% is considered to be high load to a degree but also medium load to a lower degree. For this thesis, the fuzzy intersection operation is needed, as the rules have multiple antecedents. The fuzzy intersection operation can be defined using different operators [Men95, p. 351]. Here, the algebraic product is used:

$$\mu_{A \cap B}(x) = \mu_A(x)\mu_B(x) \tag{2.3}$$

The *interference system* uses fuzzy rules to map the fuzzy input sets to outputs. Fuzzy logic rules are `IF-THEN` rules where the input, also called antecedent, is mapped to an output, also called consequent. An example rule could be “IF workload is high and response time is high THEN scale out by 1”.

Finally, the *defuzzifier* takes the outputs of the interference system and converts them back into a crisp output. If the fuzzy rules have discrete outputs, which is the case for the rules in this thesis, a centroid defuzzifier can be used. For rule outputs y_i, the fuzzy logic system output is then calculated as follows:

$$\bar{y} = \left[\sum_{i=1}^{I} y_i \mu_B(y)\right] / \left[\sum_{i=1}^{I} \mu_B y\right] \tag{2.4}$$

where I is the number of rules, y_i is the consequent of rule i, B_i is the antecedent of rule i and $\mu_B(y)$ is the membership degree of the crisp input x and fuzzy set B.

3 Related Work

State-of-the-art autoscaling approaches and comparisons of them are presented in the related work section. It also describes how this thesis differs from the existing work.

There is a large body of research on elasticity and scaling in the cloud including several surveys classifying different approaches [CBY19; GMP+21; LML14; SGJN19; VB21]. However, these comparisons mainly focus on the advantages and drawbacks of each individual category of approaches and high-level differences such as implementation difficulty or general performance. A more concrete comparison between the prediction quality of different approaches on the same or similar workloads is often not provided.

A recent paper that compares different reinforcement learning-based scaling approaches is "Integrating OpenAI Gym and CloudSim Plus" by Agos Jawaddi and Ismail [AI24]. It integrates the OpenAI gym environment into the CloudSim Plus architecture simulator and then uses this integration to implement three different deep reinforcement learning-based scaling approaches using Deep Q-Networks, Double Deep Q-Networks, and Proximal Policy Optimization. The learning speed of these agents and the performance of the system resulting from the scaling decisions are evaluated. However, the scaling is limited to the vertical scaling of a VM in relation to its CPU utilization. In contrast, this thesis will support horizontal scaling and the modeling of scaling approaches such that they can be used with the Slingshot approach.

The approaches are typically implemented either in the real world or using the CloudSim toolkit [BRC09] or some derivative. However, the model-creation process for CloudSim is much more involved compared to Slingshot, as the CloudSim model behavior needs to be modified by coding in Java compared to the quicker model-creation for Slingshot, which does not use a general-purpose programming language [KKSB23].

Thus, the goal of this thesis of supporting predictive scaling in the Slingshot approach has the novelty of fully including learning-based approaches on the modeling side, such that architects, analysts, and domain experts can quickly model scalable software architectures that include predictive, learning-based scaling approaches similarly to the approaches used in the real world. In contrast to cloud simulators such as CloudSim, where predictive scaling has already been implemented, this thesis will extend the SPD modeling language, which is fundamentally based on policy-driven management, with support for these predictive scaling approaches. This extension thus presents a novel way of abstracting learning-based resource management policies and integrating them into an existing scaling policy framework.

4 Literature Review

In this chapter, related works that were discovered during the literature review will be discussed. This includes the approaches that were selected for implementation in the Slingshot simulator. First, the methodology of the performed literature review is explained in Section 4.1. The criteria chosen for the selection of approaches will be discussed in Section 4.2. Then, the discovered approaches are discussed and classified in Section 4.3. Finally, Section 4.4 provides a rationale for the approaches that were selected for implementation. This chapter thus answers **RQ1.1**: Which of the state-of-the-art learning-based elasticity policies should be selected for modeling and simulation?

4.1 Review Methodology

As there are many papers concerning different approaches for scaling cloud applications, some strategies were used to limit the number of papers that needed to be screened. Because of this, the review process is inspired by the methodology of rapid reviews as proposed by Cartaxo et al. [CPS20]. Therefore, the different components of a rapid review and how they are performed will be discussed in the following sections.

4.1.1 Search Strategy

In order to limit the search space, three literature reviews on cloud autoscaling systems were used as a starting point. The selected surveys are "A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments" by Lorido-Botran et al. [LML14], "A Survey and Taxonomy of Self-Aware and Self-Adaptive Cloud Autoscaling Systems" by Chen et al. [CBY19] and "Reinforcement Learning-Based Application Autoscaling in the Cloud" by Garí et al. [GMP+21]. They differ in multiple ways and thus include different approaches. Firstly, they were published at different times. Secondly, they have different foci, i.e. [GMP+21] is focused on Reinforcement Learning (RL) while [CBY19] is especially concerned with the classification of different approaches regarding different criteria such as the levels of self-awareness, the used architecture patterns, the granularity of control, or the used quality modeling approaches.

Thus, these three surveys should provide a broad overview of autoscaling approaches.

4.1.2 Selection Procedure and Quality Appraisal

As these surveys mention over 100 approaches, there need to be some selection criteria to further narrow down the number of papers that will be screened.

To achieve this, the classification provided by the surveys is utilized. Consequently, only the approaches that meet the criteria for a Slingshot implementation will be evaluated. The criteria used for this depend on the used survey.

For [LML14], only the approaches that are classified as predictive are evaluated, as Slingshot already supports reactive scaling and the goal of this thesis is the addition of predictive elasticity policies. Filtering the approaches based on whether they support horizontal or vertical scaling was also considered, as Slingshot only supports horizontal scaling. However, as the approaches that only support vertical scaling could feasibly be modified to support vertical scaling, this classification was not used for filtering.

For [CBY19], the filtering was more difficult as the provided classification does not show which approaches have predictive qualities. The considered approaches are those that are either classified as control-theoretic approaches or as approaches based on implicit search-based optimizations, as those typically provide a limited implementation complexity and sometimes try to proactively or predictively scale the resources.

Finally, all the approaches that are classified as solving the scaling problem by [GMP+21] were also considered, while the approaches that solve the scheduling problem were not considered.

Quality appraisal was only sparingly performed. It was assumed that the surveys that are used as part of the selection procedure already did some quality appraisal themselves. In addition, the evaluation of the approaches is considered as a criterion. Here, the quality of the evaluation was also considered, as described in more detail in Section 4.2.1.

4.1.3 Extraction and Synthesis Procedure

The extraction procedure was carried out solely by the main author of this thesis, which may introduce personal biases, particularly regarding more subjective selection criteria. Potential biases are discussed in further detail for each criterion in Section 4.2. If any data was missing, the study was still included in the evaluation, but the missing data point was marked.

The synthesis procedure is focused on identifying the approaches that are particularly relevant for Slingshot, as the rapid review was conducted in the context of the search for a fitting approach for inclusion in the Slingshot approach. As such, narrative synthesis is used as knowledge synthesis procedure.

4.2 Selection Criteria

In order to compare the approaches and to finally select those approaches for implementation that fit the Slingshot approach, several criteria are considered and evaluated for each of the considered approaches. The criteria are shown in Figure 4.1. The figure also describes how the different criteria are weighted to get to a final decision about which approach to integrate into Slingshot. Each criterion is described in detail in the following sections.

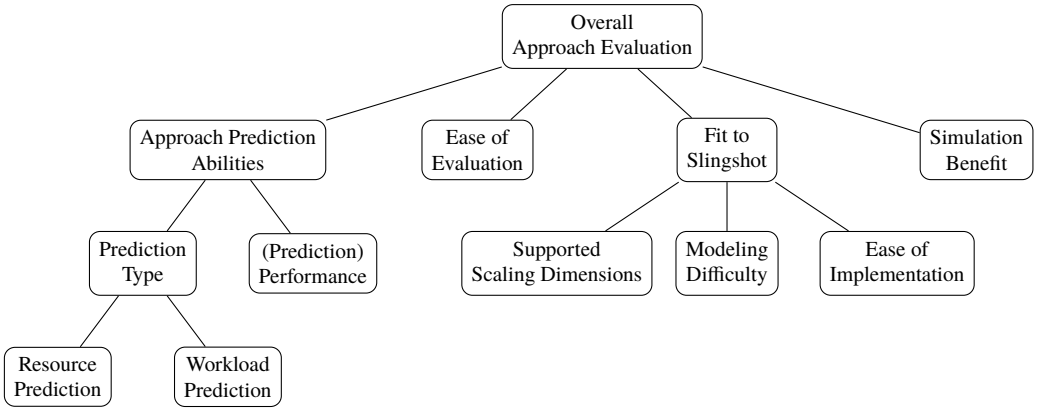


Figure 4.1: Classification of the Selection Criteria

4.2.1 Prediction Abilities

This first category of criteria is focused on the predictive qualities of the classified approaches. As reactive elasticity policies are already supported by Slingshot, we search for approaches that are predictive. The given criteria distinguish between different types of prediction and try to classify the prediction performance of the approaches.

Classification of Prediction

Some existing surveys such as Lorido-Botran et al. already make a distinction between reactive and predictive approaches (sometimes also called proactive approaches) [LML14]. However, when looking further into the different papers that are labeled as predictive by Lorido-Botran et al., several types of predictions can be distinguished.

The first type of prediction is workload prediction. The core problem that is solved by workload prediction is to predict how the workload, which can be measured in requests per time unit, will change in the future given the current workload and optionally a history of previous workloads. Workload prediction can be typically found in the approaches classified by [LML14] as time series analysis.

Another distinct type of prediction is resource prediction. For a given workload, resource prediction will predict how many resources, e.g., CPU cores, are needed to fulfill all the requests of the workload or give a prediction of the response times given some resources and a workload. Resource prediction is thus related to the problem that is described as performance modeling by [LML14].

The distinction between these two types of prediction can be fuzzy sometimes, as some approaches such as those based on reinforcement learning models often do not distinguish between them and instead often use their approach to both try to anticipate future demand and to predict the needed resources. However, this distinction still can be valuable as some approaches only perform one type of prediction. Due to the difficulty of distinguishing which type of prediction each approach employs, the classification is not binary but there is instead a third option for approaches that cannot be clearly identified. This third option is also used for approaches that do not explicitly perform a type of prediction but for which it could be argued that the prediction is still implicitly done. An

Approaches	Workload Prediction
[APJE17; BD19; BHD13; CCG13; DKM+11; DMM+10; DS18; KCH09; NLV+19; RBX+09; RBXW11; SHE13; TJDB06; WGOI17; WKL+19; XJC13; XRB12]	-
[AKTE12; ATE12; BGS+09; CDM10; CDM11; CGS03; CHL+08; DGVV12; FLWC12; GJP18; HA18; HLY12; IKLL12; JPLC13; KGM10; LZ10; MBE13; MBL+13; MWY+10; PHS+09; PN09; RDG11; SSGW11; USC+08; WXZ+11; YLS+14; ZA12; ZXW10]	✔
[BVU+10; CH11; CS13; GT12; IDCJ11; LPD+12; MMZV13; SSSS11; ZCS07]	✗

Table 4.1: Workload Prediction Ability of Approaches

example of this can be approaches based on reinforcement learning models that use the current amount of resources and the current workload as input and try to predict the needed resources for the next time unit. These approaches quite clearly try to do resource prediction, as they try to correlate a workload to a resource requirement. However, it is hard to evaluate whether there is any workload prediction involved. Based on the single workload datapoint provided to the model, a true prediction of future workloads as done by regression models is not possible, however, combined with the current resources, a rudimentary workload prediction could be done. Thus, these models are not classified as doing workload prediction nor as not doing it.

As the criteria extraction was performed solely by the main author, there might be implicit biases involved. This is especially true for approaches that do not clearly fit into a prediction class. To avoid biases as much as possible, the classification into the true or false classes was only performed for clear cases.

The results of the classification can be seen in Table 4.1 for workload prediction and Table 4.2 for resource prediction, where a checkmark (✔) is used to signify support for a type of prediction while a cross (✗) is used to signify that a approach does not support a type of prediction. For the other approaches (-), no clear classification could be performed. Most of the evaluated approaches perform resource prediction, while fewer approaches employ workload prediction.

Prediction Performance

The prediction performance criterion classifies each approach based on the results of the performed evaluation. An approach that is selected for the Slingshot approach should provide satisfactory prediction performance such that it can be used for a realistic evaluation of the given architecture models.

In addition to looking at how well an approach performs for the given evaluation, the evaluation itself is also critiqued. This is thus the only criterion where some quality appraisal is performed. It is evaluated whether the used evaluation tries to capture realistic scenarios, ideally using a real

Approaches	Resource Prediction
[HLY12; JPLC13; MWY+10; PN09; SSGW11; ZXW10]	-
[AKTE12; APJE17; ATE12; BD19; BGS+09; BHD13; BVU+10; CCG13; CGS03; CH11; CHL+08; DGVV12; DKM+11; DMM+10; DS18; FLWC12; GJP18; GT12; HA18; IDCJ11; KCH09; LPD+12; LZ10; MBE13; MBL+13; MMZV13; NLV+19; PHS+09; RBX+09; RBXW11; RDG11; SHE13; TJDB06; WGOI17; WKL+19; WXZ+11; XJC13; XRB12; ZA12; ZCS07]	✔
[CDM10; CDM11; CS13; IKLL12; KGM10; SSSS11; USC+08; YLS+14]	✗

Table 4.2: Resource Prediction Ability of Approaches

workload or trying to reconstruct the characteristics of realistic workloads. It is also evaluated how long the approach needs to perform well if it is learning-based and whether any other approaches claim to have superior performance in their own evaluation.

The approaches are then grouped into several categories. Approaches that do not perform a sufficient evaluation or perform an evaluation on a problem that is orthogonal to scaling are categorized as *N/A*. Approaches that either perform worse than other approaches in their own evaluation or in evaluations in other papers are grouped in the category *worse than other approaches*. The category *missing details* encompasses approaches that do not describe the evaluation in sufficient detail. Finally, the rest of the approaches are either classified as *good* or *okay* based on the quality of the performed evaluation and the performance.

The results of the performed classification can be seen in Table 4.3. Most of the evaluated approaches have either *good* or *okay* performance.

4.2.2 Ease of Evaluation

The ease of evaluation criterion approximates if the described evaluation of an approach could be replicated inside Slingshot. As the comparison of the performance of an approach inside Slingshot and in the original evaluation environment is a part of this thesis, this criterion should ensure that this comparison will be possible. The evaluation replication is rated as *not possible* (✗) for approaches that either do not describe the evaluation in sufficient detail, do not provide the workload data used for the evaluation or that use workload data or benchmarks that are proprietary or no longer available. Approaches for which the evaluation can most definitely be replicated are marked as *possible* (✔). For the rest of the approaches, there might be some difficulty in replicating their evaluation inside Slingshot, e.g., if the original benchmarks are no longer easily available or if some detail is a bit fuzzy, but could probably be replicated close enough. These approaches are classified into the third category (-).

The classification for this criterion can be seen in Table 4.4. Most of the classified approaches can either definitely or most likely be evaluated using Slingshot.

Approaches	Prediction Performance
[APJE17; ATE12; BD19; DGVV12; DMM+10; DS18; GJP18; HA18; LZ10; MBE13; MBL+13; MWY+10; NLV+19; RBXW11; SSGW11; XJC13; YLS+14; ZA12; ZCS07; ZXW10]	good
[GT12; SHE13]	missing detail
[BVU+10; CH11; CS13; FLWC12; HLY12; KGM10; LPD+12; MMZV13; PN09; SSSS11]	N/A
[AKTE12; BGS+09; BHD13; CCG13; CDM10; CDM11; CGS03; IKLL12; JPLC13; KCH09; PHS+09; RBX+09; RDG11; TJDB06; USC+08; WXZ+11; XRB12]	okay
[DKM+11; WGOI17; WKL+19]	slow training/convergence
[CHL+08; IDCJ11]	worse than other approaches

Table 4.3: Prediction Performance of different approaches

Approaches	Ease of Evaluation
[BHD13; DKM+11; GJP18; HLY12; IDCJ11; KGM10; LPD+12; LZ10; MMZV13; PHS+09; RBXW11; RDG11; SHE13; SSGW11; TJDB06; USC+08; XRB12; YLS+14; ZA12]	-
[AKTE12; APJE17; ATE12; BD19; CDM10; CGS03; CS13; DMM+10; DS18; HA18; KCH09; MBE13; MWY+10; NLV+19; RBX+09; WGOI17; WKL+19; WXZ+11; XJC13; ZXW10]	✔
[BGS+09; BVU+10; CCG13; CDM11; CH11; CHL+08; DGVV12; FLWC12; GT12; IKLL12; JPLC13; MBL+13; PN09; SSSS11; ZCS07]	✗

Table 4.4: Ease of Evaluation of Approaches

4.2.3 Fit to Slingshot

Whether an approach fits into the Slingshot approach depends on multiple factors. First, the supported scaling dimensions of the approach need to be considered, as the Slingshot approach only supports horizontal scaling. Another consideration is the SPD metamodel changes that would be needed to integrate an approach into Slingshot. Finally, the ease of implementing an approach is also evaluated.

Approaches	Scaling Dimension
[AKTE12; APJE17; ATE12; BD19; BGS+09; BHD13; CCG13; CS13; DKM+11; DMM+10; DS18; GJP18; GT12; HA18; IDCJ11; JPLC13; LZ10; MBE13; MBL+13; MWY+10; NLV+19; PN09; RDG11; USC+08; WGOI17; WKL+19]	horizontal
[CDM11; KCH09; PHS+09; RBX+09; RBXW11; SSGW11; WXZ+11; XJC13; XRB12; ZA12; ZXW10]	vertical
[DGVV12; FLWC12; KGM10; YLS+14]	horizontal and vertical
[BVU+10; CDM10; CGS03; CH11; CHL+08; HLY12; IKLL12; LPD+12; MMZV13; SHE13; SSSS11; TJDB06; ZCS07]	none

Table 4.5: Scaling Dimensions supported by the Approaches

Supported Scaling Dimension

This criterion evaluates if an approach includes the description of a scaling mechanism and whether that mechanism scales the system horizontally, e.g., by increasing the CPU core count or RAM amount, or vertically, i.e., by adding new instances running a service. Approaches that additionally consider scaling using heterogeneous resources or describe reconfiguration mechanisms are listed as supporting horizontal scaling.

The broad classification of all approaches is provided in Table 4.5. As only approaches that can scale horizontally were considered for some of the surveys, most of the classified approaches support horizontal scaling. Some approaches support both horizontal and vertical scaling. Some other approaches do not support scaling at all and were thus classified in the *none* category.

Modeling Difficulty

The Modeling Difficulty criterion estimates the SPD model changes that would be needed to integrate an approach into Slingshot.

For example, for approaches that use cloud costs as a training parameter for learning-based approaches or as part of a cost function, this cost would need to be somehow modeled or replaced if the approach should be integrated into Slingshot, as Slingshot does not currently model cloud costs. These approaches are thus classified in the *cost* category.

The *metric detail* category encompasses approaches that would need some more detailed metrics than currently provided by the Slingshot approach, while Approaches in the *no scaling* category do not support scaling.

Some approaches are also focused on scientific batch workloads or other system models. Slingshot should however be able to be used for general purposes and for modeling typical enterprise cloud systems. Thus, these approaches are classified as posing *architecture* modeling challenges.

It is additionally evaluated if the approach fits into slingshot when modeled. Approaches that need a lengthy calculation for each reconfiguration do also not fit and are classified as *too slow*.

Approaches	Modeling Difficulties
[AKTE12; BVU+10; CDM10; CS13; DS18; PN09; TJDB06; USC+08]	architecture
[BHD13; DGVV12; DKM+11; JPLC13; NLV+19; PHS+09; SHE13; WGOI17; WKL+19]	cost
[CGS03; RBX+09; RBXW11; XRB12]	metric detail
[CH11; HLY12; IKLL12; KGM10; LPD+12; MMZV13; SSSS11; ZCS07]	no scaling
[APJE17; ATE12; BD19; BGS+09; CCG13; CDM11; CHL+08; DMM+10; FLWC12; GJP18; GT12; HA18; IDCJ11; KCH09; LZ10; MBE13; MBL+13; RDG11; SSGW11; WXZ+11; XJC13; YLS+14; ZA12; ZXW10]	okay
[MWY+10]	too slow

Table 4.6: Difficulty of including Approaches into the SPD Model

Approaches that do not pose any significant Modeling Difficulty are classified into the *okay* category.

The classification is shown in Table 4.6. The main difficulties that could be faced when integrating Approaches into the SPD model are either architectural or based on the missing cost modeling of Slingshot. There are also quite a few approaches that do not explicitly consider scaling. The largest category is however the category of approaches that can be modeled in Slingshot without any significant difficulties.

Ease of Implementation

The Ease of Implementation criterion describes how feasible the implementation of an approach is. This is decided both by whether the provided description of the approach contains sufficient detail and by the approximated complexity of the approach. The complexity of the approach is mainly approximated by looking at whether different models or algorithms that interact with each other are used. It is also evaluated if external components such as an integer linear program solver would be required for an approach implementation. Additionally, the personal experience of the author of this thesis with the used methods, such as reinforcement learning, genetic optimization, or regression, is also included in the rating. The approaches are then classified into either the *difficult* or *possible* category.

Learning-based approaches that require a large training dataset are also rated as hard to implement, as there is no available large training dataset for Slingshot. These approaches are assigned to the *offline training* category, as they would need the aforementioned dataset and some offline training.

Approaches	Ease of Implementation
[BVU+10; CCG13; CGS03; DGVV12; FLWC12; JPLC13; KCH09; LZ10; MWY+10; PN09; SHE13; SSSS11; TJDB06; ZA12]	difficult
[GT12; KGM10]	missing details
[CDM10; CDM11; CH11; HLY12; IKLL12; LPD+12; MMZV13; ZCS07]	no scaling
[BGS+09; CHL+08; RBXW11; USC+08; XRB12]	offline training
[AKTE12; APJE17; ATE12; BD19; BHD13; CS13; DKM+11; DMM+10; DS18; GJP18; HA18; IDCJ11; MBE13; MBL+13; NLV+19; PHS+09; RBX+09; RDG11; SSGW11; WGOI17; WKL+19; WXZ+11; XJC13; YLS+14; ZXW10]	possible

Table 4.7: Ease of Implementation for different approaches

Finally, the *missing details* category encompasses the approaches that are not described in sufficient detail while the *no scaling* category contains all approaches that do not describe how scaling should be implemented or are orthogonal to the scaling problem.

Because of this and the general difficulty of evaluating the ease of implementation for any given algorithm, this is not meant to be an objective criterion but rather a subjective measure of the author that tries to further distinguish between the approaches specifically for the context of this thesis and the inclusion of an approach inside the Slingshot Approach.

The categorization of the approaches is given in Table 4.7

4.2.4 Simulation Benefit

The implementation of an elastic policy approach inside the Slingshot Approach might not only serve to enrich the functionality of the Slingshot Approach but can also help to improve the elastic policy approach. All the benefits enabled by the implementation of an elastic policy approach inside the Slingshot approach are evaluated for the Simulation Benefit criterion.

Some of the simulation benefits could be listed for each approach. For example, the created model for the elastic policies could be used for code generation such that a Slingshot model can be transformed into working code including the selected scalability approach. Additionally, the addition of any elastic policy approach to the Slingshot Approach makes it easier to perform comparisons between different scalability approaches that are implemented inside Slingshot for a modeled application. Depending on the characteristics of an application, different elastic policy approaches might perform differently and thus evaluating different elastic policy approaches for a modeled system can help to find the optimal approach for that specific system.

There are however also some benefits of modeling an elastic policy approach that only applies to some approaches, and only those benefits are listed in Table 4.8. These individual benefits include *parameter tuning* that can be performed more easily and quickly when using the approach

Approaches	Simulation Benefit
[BD19; BHD13; BVU+10; CH11; CHL+08; CS13; DKM+11; DS18; FLWC12; GJP18; GT12; HLY12; IDCJ11; IKLL12; JPLC13; KCH09; KGM10; LPD+12; MBE13; MMZV13; MWY+10; NLV+19; PN09; RBX+09; SHE13; SSGW11; SSSS11; TJDB06; USC+08; WKL+19; XJC13; XRB12; ZA12; ZCS07]	/
[APJE17; DMM+10; HA18; LZ10; RBXW11; WGOI17; WXZ+11]	model transfer
[AKTE12; ATE12; CCG13; CDM10; CDM11; CGS03; DGVV12; PHS+09; RDG11; YLS+14; ZXW10]	parameter tuning
[BGS+09; MBL+13]	simulation

Table 4.8: Simulation benefit for different Approaches

modeled inside the Slingshot simulator as opposed to running the elastic policy approach on a real testbed. This benefit is only listed for approaches that have many parameters or explicitly call out the importance of parameters and their tuning in the corresponding paper.

Another individual benefit relates to learning-based models that need to be trained on some data. As shown in [WGOI17], a learning-based model that is trained inside some simulator can be transferred to a real cloud system in order to enhance the initial performance of a learning-model-based approach in the real world. The mentioned paper only looks at the transfer of a model trained in CloudSim, but similar results can probably be obtained using a model trained using the Slingshot Approach. Approaches are classified as benefiting from *model transfer* if they mention or evaluate this possibility in their paper or if the output of a model could be used to create non-learning-based policies, e.g., if the model learns thresholds that can be used without the learning model.

Finally, the *simulation* simulation benefit is given to approaches that utilize some simulation themselves. For these, this internal simulation could be replaced by Slingshot, which could be beneficial to the approach as well as to Slingshot.

4.3 Approach Overview

In this section, the approaches that fulfill the criteria listed in Section 4.2 the best are discussed. A tabular comparison of all evaluated approaches is provided in Appendix A.

Only approaches that are classified as utilizing either workload or performance prediction, scaling horizontally, and having an evaluation that is both replicable and shows good performance are considered.

4.3.1 Reinforcement Learning-based Approaches

Most of the promising approaches that try to both do performance and workload prediction and for which their evaluation can be replicated are Reinforcement Learning (RL) approaches taken from [GMP+21]. They mostly differ in their approach to modeling the autoscaling problem as an RL model.

Most of the RL approaches use one or more workload indicators such as the number of requests and at least one resource indicator such as the allocated VM count as part of the state model [APJE17; DMM+10; DS18; WGOI17; WKL+19]. Extensions to this model include the modeling of different types of VMs [WKL+19], the monitoring of metrics per server and per application [NLV+19] or per task [BD19], and the inclusion of the rate of change for metrics in the state [HA18].

The other important aspect of formulating an RL model is the action space. The most straightforward modeling simply allows the actions to remove or add a specified number of VMs [APJE17; DMM+10; DS18; HA18; WGOI17]. Some approaches also consider different types of VMs [BD19; WKL+19] or add additional actions such as merging and moving applications [NLV+19]. A completely different modeling approach is mentioned by Horovitz and Arian. Their Q-Thresholds model tunes the thresholds that are also used for reactive scaling and has actions for increasing or decreasing the thresholds [HA18].

Finally, RLs also need a reward for training. The reward functions are usually either based on the observed response times in comparison to a given Service Level Agreement (SLA) and utilization [DMM+10; HA18] or based on some cost model, either in combination with penalties for high utilization [WGOI17] or for SLO violations [APJE17; DS18]. Cost models can also penalize based on low load [DS18] or estimate the profit a system would generate [WKL+19].

An important problem for all RL-based approaches is the duration it takes for the model to learn the environment. Several techniques have been proposed to either reduce the state space or otherwise accelerate the learning of the RL model. Arabnejad et al. use Fuzzy Membership functions in order to reduce the state space and accelerate the learning [APJE17]. The Q-Thresholds model proposed by Horovitz and Arian reduces the state space by only considering the number of VMs and the action space by only providing two actions which are interpreted depending on the last scaling action taken by the system [HA18] while the Improved Q-Learning method reduces the state space by using an interval-based mapping instead of a discrete state space and the action space by using a sliding window of possible actions.

4.3.2 Other Approaches

Some other approaches also fit the criteria but are not based on reinforcement learning.

Ali-Eldin et al. propose a control theory-based approach that models the system as a set of queues and estimates the needed resources for the next timestep based on the previous load and different gain parameters that either represent the periodical rate of change of the system load or the ratio between the change in the load and the average system rate over time [ATE12]. This proactive controller is combined with a reactive controller in different ways, i.e. only using the predictive

controller for upscaling but not for downscaling. A follow-up to this paper uses a similar model but focuses on scientific workloads that can be buffered [AKTE12]. It is thus less interesting for the Slingshot approach which should also be usable for other cloud applications.

Mi et al. use quadratic exponential smoothing for load forecasting and then a chromosome coding-based strategy for reconfiguration [MWY+10]. However, each reconfiguration takes 28 seconds to perform [MWY+10, p. 519]. As the Slingshot simulator should be able to simulate a large timespan within a small time unit, this approach is not feasible.

Moore et al. propose a novel approach where first, a time series prediction is performed using scalable vector machines [MBE13]. This prediction is then used as an input to a first naive bayes model which predicts the likelihood of an SLO violation when a given number of servers is allocated. Another model is used to produce bounds on the number of servers needed for the predicted scaling load and to check the output of the first prediction model for plausibility. A search algorithm can then be used to find a number of servers for which SLO violations are unlikely to occur in the predicted timeframe.

4.4 Selected Approaches

Initially, it was considered to implement the approaches Improved Q-Learning and Q-Thresholds by Horovitz and Arian [HA18]. However, the author claims to have multiple patents related to these approaches [AHZ17; HAZ18a; HAZ18b]. As the applicability of these patents was unclear, an approach that is not patented was used instead.

The Fuzzy Q-learning and Fuzzy SARSA approaches by Arabnejad et al. also performed well in their evaluation [APJE17]. While they do not contain an explicit workload prediction component, the approaches are described in sufficient detail and the evaluation can be replicated. They do not pose any significant modeling difficulties and support horizontal scaling which is needed for Slingshot. The learned Q-Values are essentially fuzzy scaling rules and could thus also be used in fuzzy scaling systems without using reinforcement learning. A detailed overview of the approaches is given in Section 5.2.1.

5 Concept

This chapter describes the current SPD metamodel and how the new scaling methods differ from the existing ones in Section 5.1. Additionally, a conceptual overview of the selected approaches is given in Section 5.2. This section provides the fundamentals for answering **RQ2.1**: How and at what abstraction level can the selected approaches be effectively implemented in Slingshot?

5.1 Extending the SPD metamodel

The first step to extend the Slingshot approach is extending the SPD metamodel that is used by architects to model the scaling behavior of an application. For this, an overview of the current metamodel is given in Section 5.1.1 and the different needs for the new policies are described in Section 5.1.2

5.1.1 The current SPD metamodel

The current SPD metamodel comprises scaling policies that each have a trigger, an adjustment action, and optional constraints. Each scaling policy also contains a reference to the target group which should be affected by the adjustment actions. The trigger specifies when a scaling action should be performed. There are several kinds of triggers. Basic triggers contain two children, a stimulus that specifies which measure triggers the scaling action, and an expected value. The trigger itself then specifies how the stimulus and the expected value should be compared to each other. If a stimulus is composed of several measuring points, e.g.g, the measuring of a metric for all resource containers inside a service group, it also has a property specifying the type of aggregation that should be performed. There are also composed triggers that contain multiple basic triggers combined using a logical operator.

The adjustment action then specifies which adjustment should be performed when a trigger evaluates to true. Several adjustment types exist that modify the count of some architecture components based on some relative or absolute adjustment given at modeling time.

Finally, some constraints can modify the scaling decision made by a trigger. Policy constraints can limit when a scaling policy is allowed to perform a scaling action. Target constraints limit the scaling actions that can be performed on a target, e.g., by constraining the number of containers or limiting thrashing by only allowing opposing scaling actions after a cooldown.

5.1.2 Model-based Scaling Approaches

The model-based scaling approaches presented in this thesis differ from the existing SPD scaling policies mainly in the time when the adjustment action and magnitude are determined. For the existing scaling policy, the adjustment is determined by the adjustment action of a scaling policy. While the adjustment action can vary in magnitude based on the currently allocated container count in the case of relative adjustments, the parameters defining this variation are defined at modeling time. The adjustment direction is also fixed at modeling time for the existing adjustments. For the new approaches, the adjustment magnitude as well as the adjustment direction are determined dynamically at runtime. While there could still be relevant parameters that are defined at modeling time such as a maximum or minimum scaling action, the adjustment itself is not defined at modeling time.

This also changes the role of the trigger, i.e. there is no simple trigger that determines only whether a scaling action should occur. Instead, a model determines both whether a scaling action should occur and the direction and magnitude of the scaling action.

In order to fit this new behavior into the existing SPD metamodel, two different concepts were considered and explored:

- A new trigger and adjustment type could be added to the existing model. The trigger contains the model, while the adjustment type simply refers to the adjustment being made by the trigger.
- A new scaling policy superclass could be introduced which is extended both by a class representing the existing scaling policies and a new class for the new, model-based scaling policies. The new model-based scaling policy only contains a model instead of a model and a trigger.

The main advantage of defining a new trigger and adjustment type is the simple metamodel structure. However, for a model user, introducing a new trigger and adjustment type that only work with each other and not in combination with the existing adjustment and trigger types might lead to misunderstandings about why the model seemingly enables them to combine these different kinds of adjustments and triggers while they are not compatible with each other. By introducing two different types of scaling policies, the new policies and their different behavior of dynamically determining the adjustment based on a model output is communicated to the modeler.

The scaling model itself also poses some modeling challenges. For RL-based models, there generally needs to be some input stimuli that are used for determining the state of the RL model and the reward. Different SPD metamodel extensions were considered to determine how the reward of an RL model can be modeled:

- Modeling different reward functions as objects that can be chosen independently from the RL model
- Creating a separate metamodel that enables a reward to be freely specified at modeling time, including integration of the Service Level Objective PCM addon
- Assigning a fixed reward function for each RL model that cannot be changed

Allowing the specification of either predefined or fully customized rewards for models poses a problem because RL models sometimes have some assumptions about the behavior of a reward function. For example, for Improved Q-Learning, the reward function is based on adhering to an SLO and is assumed to be negative whenever the given SLO is violated and positive whenever it is met, with the positive value declining as more resources are used up [HA18]. For a reward function that does not fulfill this assumption, the model will not work as expected. Thus, when allowing custom reward functions, the problem of the modeler using reward functions and models that are not compatible arises. This could result in confusion and a bad user experience if no validation of the reward function and detailed error messages are provided. As the models discussed in Chapter 4 typically specify the used reward function in sufficient detail, it was chosen to associate a fixed reward function with each model. Some of the parameters used in the reward function and the used inputs can however still be chosen, allowing some adaptation of the reward function by the modeler. Additionally, sensible defaults for the parameters that are taken from the corresponding papers are set or determined experimentally, such that the models should work without any manual tuning. This allows a modeler to easily compare different model types and their performance impact, as only the model object and the required inputs need to be changed. This also mostly hides the reward function complexity from the modeler.

5.1.3 Stimuli model extension

The SPD metamodel is also extended with regard to the stimuli. Currently, the aggregation of stimuli such as the CPU state is limited to traditional aggregation functions such as the average, minimum, maximum, median, and sum. However, learning-based models often also have different aggregation needs. Often, the reward functions are based on adherence to an SLO. As SLOs often specify the required response time in percentiles [DCS+19], a 95th percentile aggregation method that outputs the 95th percentile value of the aggregated variable is added.

5.2 Using Reinforcement Learning for determining scaling actions

In Section 4.4, the approaches selected for implementation in the SPD metamodel and the Slingshot simulator were discussed. In this chapter, they are described in further detail and the changes that were made compared to the original papers are described.

There are different kinds of learning models. A reinforcement learning model, such as Fuzzy Q-learning, typically operates periodically. It collects measurements to determine both the current state and the inputs needed for the reward function. After gathering this information, the model periodically updates the Q-values and evaluates which scaling decision should be made based on the current state.

Some models not only evaluate the action value function periodically and perform actions based on the RL policy but also perform scaling actions during these intervals. An example of this is the Q-Thresholds model presented in [HA18]. The periodical update and evaluation affect thresholds which are used to do scaling actions in between these periodical evaluations. This model scaling behavior is classified as event-based.

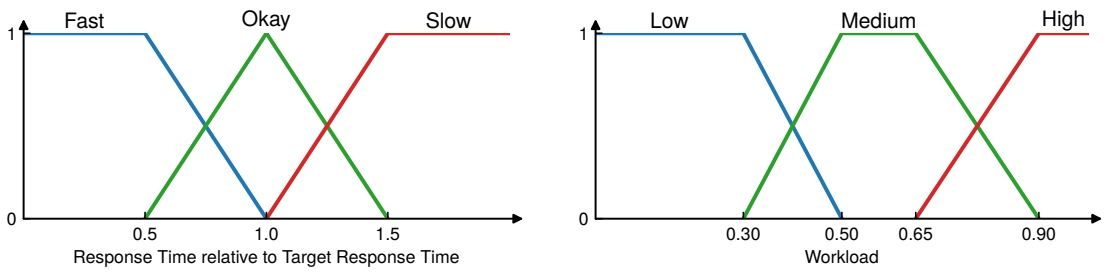


Figure 5.1: Fuzzy membership functions

Response Time	Low Workload					Medium Workload					High Workload				
	-2	-1	0	1	2	-2	-1	0	1	2	-2	-1	0	1	2
Fast	0.2	0.1	0	0	0	0.1	0.2	0.1	0	0	0	0.1	0.2	0.1	0
Adequate	0.1	0.2	0.1	0	0	0	0.1	0.2	0.1	0	0	0	0.1	0.2	0.1
Slow	0	0.1	0.2	0.1	0	0	0	0.1	0.2	0.1	0	0	0	0.1	0.2

Table 5.1: Action values for knowledge-based initialization

5.2.1 Fuzzy Q-Learning and Fuzzy SARSA

The Fuzzy Q-learning and Fuzzy SARSA methods both use Reinforcement Learning, which is described in more detail in Section 2.2, and Fuzzy Logic, which is described in Section 2.3. The implementation is based on the work by Arabnejad et al. [APJE17]. The Fuzzy Q-learning method was first described by Glorennec and Jouffe [GJ97] and applied to application scaling by Jamshidi et al. [JSP+15].

The algorithms consist of several parts:

1. First, the action values are initialized. They can either be initialized pessimistically (with value 0), optimistically (with value 1), or with some predefined knowledge. The knowledge initialization used in this thesis is shown in Table 5.1.

2. The current state is recorded. For Fuzzy Q-Learning, a state consists of the resource utilization in the last period and the response time while the state for Fuzzy SARSA also contains the number of allocated containers.

$$s_{\text{SARSA}} = (\text{utilization}, \text{response time}, \text{container count})$$
$$s_Q = (\text{utilization}, \text{response time})$$

For each of these states, there are also corresponding fuzzy states:

$$\tilde{s}_{\text{fuzzy(SARSA)}} = (\text{fuzzy utilization}, \text{fuzzy response time}, \text{container count})$$
$$s_{\text{fuzzy}(Q)} = (\text{fuzzy utilization}, \text{fuzzy response time})$$

The used fuzzy membership functions are trapezoidal and shown in Figure 5.1.

3. Next, for each fuzzy state, a partial action is selected while using an epsilon-greedy policy.

$$a_{\text{fuzzy}}(s_{\text{fuzzy}}) = \begin{cases} \text{argmax}_a Q(s_{\text{fuzzy}}, a) & \text{with probability } 1 - \varepsilon_i \\ \text{rand}(-2, 2) & \text{else} \end{cases}$$

4. Then, a control action is calculated by calculating a weighted sum of the partial actions of the fuzzy states. The control action specifies which action the system should perform.

$$a_{\text{control}} = \sum_{s_{\text{fuzzy}}} a_{\text{fuzzy}}(s_{\text{fuzzy}}) \cdot \mu_{s_{\text{fuzzy}}}(s_i)$$

   where $\mu_{s_{\text{fuzzy}}}(s_i)$ is the fuzzy membership value as described in Section 2.3.

5. The action value for the current state and action are approximated. As the system is in multiple fuzzy states at once, the action value is approximated by calculating a weighted sum of the action values for these fuzzy states.

$$\widetilde{Q}(s_i, a) = \sum_{s_{\text{fuzzy}}} Q(s_{\text{fuzzy}}, a_{\text{fuzzy}}(s_{\text{fuzzy}})) \cdot \mu_{s_{\text{fuzzy}}}(s_i)$$

6. The calculated control action a_{control} is performed in the system.
7. After the action is taken, a new state s_{i+1} is recorded (analogously to step 1).
8. Additionally, the reward and the state value (for Q-learning) of the new state are calculated.

$$V(s_i) = \sum_{s_{\text{fuzzy}}} \mu_{s_{\text{fuzzy}}}(s_i) \cdot \text{argmax}_a Q(s_{\text{fuzzy}}, a)$$

9. The error signal is calculated. For Fuzzy Q-Learning, it is as follows:

$$e_{Q(i)} = r_i + \gamma \cdot V(s_{i+1}) - \widetilde{Q}(s_i)$$

   For SARSA, the error signal is calculated like this:

$$e_{\text{SARSA}(i)} = r_i + \gamma \cdot \widetilde{Q}(s_{i+1}, a_i + 1) - \widetilde{Q}(s_i, a_i)$$

10. Finally, the action values for the observed fuzzy states are updated using the error signal:

$$Q_{i+1}(s_{\text{fuzzy}}, a_{\text{fuzzy}}(s_{\text{fuzzy}})) = Q_i(s_{\text{fuzzy}}, a_{\text{fuzzy}}(s_{\text{fuzzy}})) + \alpha \cdot e_i \cdot \mu_{s_{\text{fuzzy}}}(s_i)$$

The described steps are combined differently in order to perform the Fuzzy SARSA algorithm (see Algorithm 5.1) or the Fuzzy Q-Learning algorithm (see Algorithm 5.2). As described in Section 2.2, SARSA differs from Q-Learning in being on-policy. Additionally, the Fuzzy SARSA algorithm also differs from the Fuzzy Q-Learning algorithm in the different state spaces, where SARSA additionally contains the number of containers. Thus, the state space for Fuzzy SARSA is larger, which could lead to needing a longer learning period. The advantage of including the number of containers in the state space is that for a different amount of containers, different scaling actions might be preferred even when the load and response time are the same. For example, it could be advantageous to scale

Algorithm 5.1 Fuzzy SARSA algorithm

```
procedure FUZZY SARSA
    Initialize action values Q
    Choose partial actions
    Calculate a control action a
    Approximate the action value for the current fuzzy state s_i and action a
    while true do
        Observe the reward
        Calculate the error signal
        Update the action values
        Perform the chosen action a
        Choose partial actions (for the next state s_{i+1})
        Calculate a new control action a
        Approximate the action value for the current fuzzy state and action
    end while
end procedure
```

Algorithm 5.2 Fuzzy Q-Learning algorithm

```
procedure FUZZY Q-LEARNING
    Initialize the action values Q
    while true do
        Choose partial actions
        Calculate the control action a
        Approximate the action value for the current fuzzy state s_i
        Take the action a and go for the next state
        Observe the reinforcement signal
        Calculate the value of the new state
        Calculate the error signal
        Update the action values
    end while
end procedure
```

out two units at a time when encountering a high utilization and response time while the container count is high, as scaling out by one container if there are already many containers might not have a big impact. Smaller scaling actions might be preferable for small container counts.

The implementation of the Fuzzy Q-Learning and Fuzzy SARSA methods in Slingshot differs from the originally described implementation in several ways. First, an exponential decay with a floor is chosen for the epsilon values. The formula used for this is the following:

$$\varepsilon_i = \begin{cases} e^{-\varepsilon * i} & \text{if } e^{-\varepsilon * i} > 0.1 \\ 0.05 & \text{else} \end{cases} \tag{5.1}$$

where ε is a user-defined value that is contained in the model. Like this, exploration is encouraged in the earlier phase where the approximation of the action values might still be erroneous, while there is much more exploitation in the later phase when the action values should be more accurate.

This was inspired by the strategy S1 discussed in [JSP+16, pp. 75–77], which converged quickly to a good policy but also was effective for highly dynamic workloads. In comparison to this S1 strategy, the chosen strategy explores less as it has a minimum of $\varepsilon = 0.05$ instead of $\varepsilon = 0.2$. Additionally, the decay of ε is smooth instead of step-based.

The reward function is also modified compared to the original paper. The used reward function is as follows:

$$r_{i+1} = \begin{cases} \frac{e^{s_{\text{util}}}}{\text{Container Count}(s_i)} & \text{if responseTime}(s_{i+1}) < \text{targetResponseTime} \\ 0 & \text{if responseTime}(s_{i+1}) < \text{responseTime}(s_i) \wedge a_i > 0 \\ e^{\frac{\text{targetResponseTime} - \text{responseTime}(s_{i+1})}{\text{targetResponseTime}}} - 1 & \text{else} \end{cases} \tag{5.2}$$

A different reward function that additionally provided rewards based on the performed scaling action was also considered:

$$r'_{i+1} = r_a + r_{i+1} \tag{5.3}$$

$$r_a = \begin{cases} -\text{Container Change}(s_{i+1}) * r_{\text{in}} & \text{if Container Change}(s_{i+1}) < 0 \\ -\text{Container Change}(s_{i+1}) * r_{\text{out}} & \text{if Container Change}(s_{i+1}) > 0 \\ 0 & \text{else} \end{cases} \tag{5.4}$$

where

$$\text{Container Change}(s_{i+1}) = \text{Container Count}(s_{i+1}) - \text{ContainerCount}(s_i) \tag{5.5}$$

r_a is an added part of the reward that should further discourage scaling actions in general, as the action reward for scaling up and then down the same amount is negative. It should also help keep the container count as low as possible by giving a penalty for scaling out and a smaller reward for scaling in. However, this modified reward function was not used, as the evaluation in Section 7.2.1 shows that the model performance degrades when using the modified reward function r'.

6 Implementation

While the previous Chapter 5 described the concepts used for integrating the new scaling functionality, this chapter focuses on the concrete changes made on a metamodel and slingshot implementation level, showing which parts of the metamodel and the simulator were altered and how. The section thus answers the research question **RQ2.1**: How and at what abstraction level can the selected approaches be effectively implemented in Slingshot? The implementation artifacts, i.e. the changed SPD metamodel and Slingshot simulator, are provided on GitHub[1].

6.1 SPD Metamodel changes

While Section 5.1 describes the newly introduced concepts needed for supporting learning-based models in the SPD metamodel, this section shows the detailed model changes that were done. The changes to the metamodel are divided into several parts. First, the reorganization of scaling policies into trigger-based and model-based scaling policies is discussed. Then, the model that is the central component for modeling the new predictive scaling policies is described. A conceptual overview of the new SPD metamodel is provided in Figure 6.1.

The SPD metamodel was also extended with new custom datatypes. The `PercentDouble` datatype is a double that contains the restriction that values need to be within 0 and 1. Using custom datatypes instead of model restrictions has the advantage of more immediate validation of the user input. For example, a user cannot input a value greater than 1 for an attribute of `PercentDouble` type when using the graphical model editor.

6.1.1 Model-based scaling policies

We introduce two different types of scaling policies: the existing policies are renamed to Trigger Based Scaling Policies. `ScalingPolicy` is now an abstract parent class extended both by the existing `TriggerBasedScalingPolicy` and the newly introduced `ModelBasedScalingPolicy`.

[1] https://github.com/berbjs-masterthesis/Palladio-Addons-SPD-Metamodel, commit ade684e4467e484ce36bb44e0465778b18ac953b
https://github.com/berbjs-masterthesis/Palladio-Analyzer-Slingshot-Extension-SPD-Interpreter, commit a7ce71230928a6a53789fed958a6b3e7d0b6029b
https://github.com/berbjs-masterthesis/Palladio-Analyzer-Slingshot-Extension-PCM-Core, commit 51dfd86b601f19a7467e0cb6839caf2f02b969f5

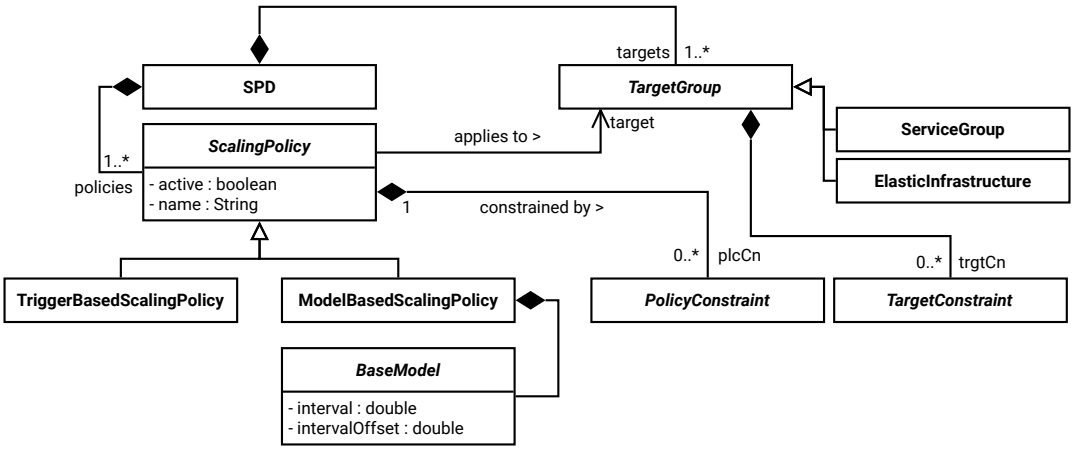


Figure 6.1: Excerpt of the new SPD metamodel

The new predictive scaling policies do not have the same structure as the existing trigger-based scaling policies. Instead of a trigger and an adjustment type, they consist of a model, which will be further described in Section 6.1.2, and an adjustment attribute that is used to store the adjustment made by the predictive model such that it can be used for a model transformation. This attribute does not need to be set by the modeler and is set by the predictive model.

The predictive model encompasses the responsibilities of both the trigger and the adjustment contained in a `TriggerBasedScalingPolicy`. However, in contrast to a trigger, the model decides not only whether an adjustment is necessary but also the magnitude of the adjustment. A significant difference is also that the models perform some actions periodically instead of performing the actions when receiving some given stimulus. Thus, the models have an `interval` attribute specifying the periodicity and an `intervalOffset` that can specify that the first interval should only begin after some amount of time has passed and thus some data has been accumulated.

6.1.2 Model

The `BaseModel` class is an abstract class representing a generic model that acts on some inputs to output some wanted adjustments to the adjustment parameter of the `ModelBasedScalingPolicy` that it is contained within. The metamodel for the model object and its children is given in Figure 6.2. The component that should be affected by the adjustment is described in the scaling policy itself. Therefore, each model class only needs to include the stimuli used as inputs and attributes that specify the parameters required for the model. In an RL-based model, such as the Fuzzy SARSA Model, inputs are stimuli that define the state or serve as feedback signals used in the reward function.

The simplest type of adjustment model is the `RandomModel` which makes a random down- or upscaling decision. It could be used to simulate the chaos engineering approach of randomly terminating virtual machine instances [BBD+16]. The attributes of the `RandomModel` class allow the modeler to choose the possible adjustment range using `minAdjustment` and `maxAdjustment` and the probability of

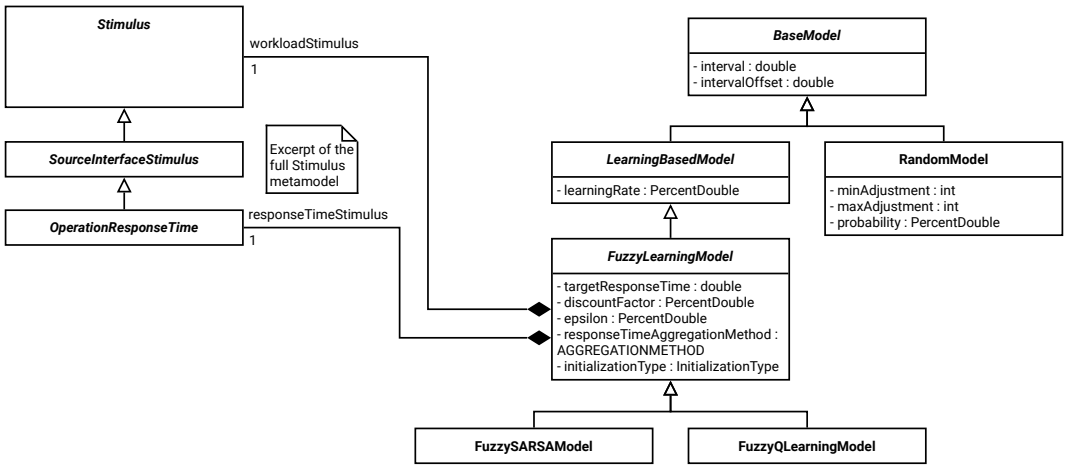


Figure 6.2: Model Metamodel

an adjustment happening using `probability`. Models that take some input and use a learning method to determine the adjustment decision are contained in the `LearningBasedModel` class. These models contain a `learningRate` attribute that specifies the learning rate used by the learning model.

The inputs and parameters for Fuzzy SARSA and Fuzzy Q-Learning models are contained in the `FuzzyLearningModel` class. The fuzzy learning models need a stimulus as an input to determine the current workload. As a wide range of stimuli could be used to represent the workload, no further restriction on which stimulus can be used was set. For example, the workload could be represented by a `CPUUtilization` stimulus, a `QueueLength` stimulus, or a `TaskCount` stimulus. Each learning model also contains a response time stimulus that needs to be of the type `OperationResponseTime`. As the response time stimulus provides discrete response time events, they need to be aggregated internally for each interval of the `FuzzyLearningModel`. The used aggregation can be chosen using the `responseTimeAggregationMethod` attribute. Additionally, the `targetResponseTime` attribute allows the modeler to choose how high the response time should be. For example, using the newly added `PERCENTILE95` aggregation method together with a `targetResponseTime` of 0.5 determines that the learning model should strive to keep the 95th-percentile response time under 0.5 seconds. The `discountFactor` and `epsilon` attributes allow the modeler to choose both the discount factor and the initial exploration probability used by the RL model.

When comparing the proposed SPD metamodel structure to the machine learning metamodel proposed by [HMS+19], several differences can be seen. Hartmann et al. propose a metamodel that contains four Blocks: An *objective*, a *learning algorithm*, an *optimizer*, and *data set metadata*. The proposed SPD metamodel however only explicitly contains the *learning algorithm* block, which similarly contains hyperparameters, i.e. the epsilon value, and an initialization method. The *data set metadata* block could be compared to the stimuli that are contained within a learning model. However, the stimuli only contain the features themselves and do not contain statistical information about the dataset. The *optimizer* is implicitly contained within the model, which also contains a learning rate parameter. The *objective* block is not needed for the SPD metamodel as the objective of the learning model is specified by the scaling policy itself - the objective is always to predict the necessary scaling actions. The differences between these metamodels come down to the target they want to achieve. While Hartmann et al. try to model machine learning in general, the SPD

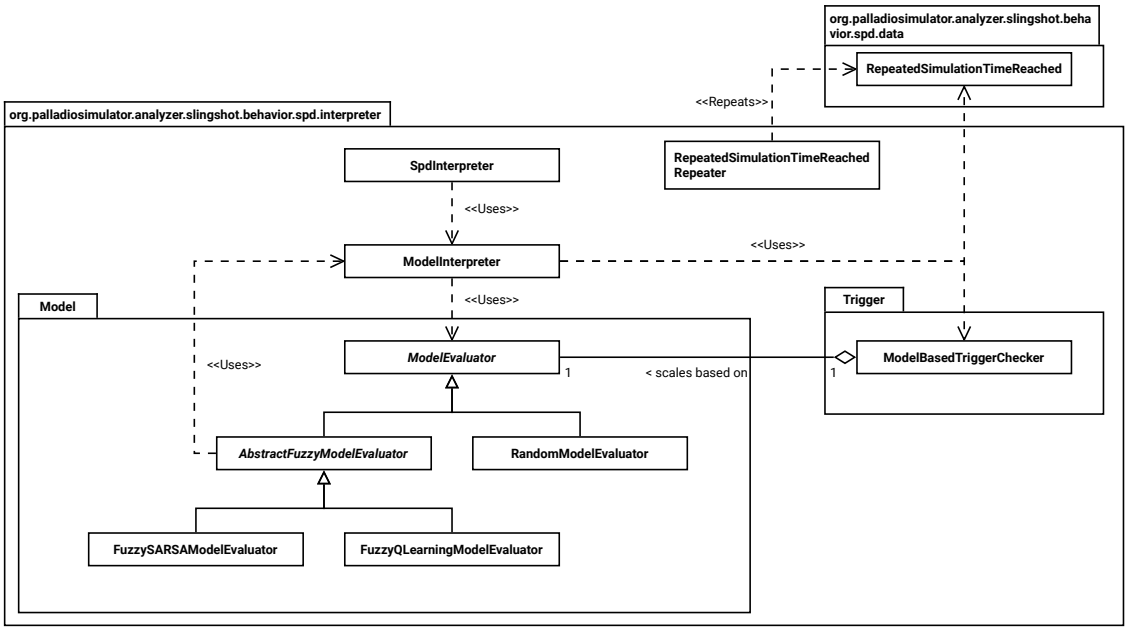


Figure 6.3: UML Class Diagram of the new classes inside the SPD interpreter package

metamodel only applies to a specific application of machine learning and tries to be easy to use for modelers. Hartmann et al. then use a meta-learning metamodel to ease the use of machine learning [HMS+19, p. 302]. It reuses the components of the machine learning metamodel, but adds new components that model a meta-learning process that selects a fitting learning model and parameters for a problem. While this meta-learning metamodel could be needed to realize the stated goal of using machine learning as a black box [HMS+19, p. 302] in the general case, the given SPD metamodel can already be used by users that have no machine learning proficiency to compare different learning models for the problem of application scalability and utilize them within the simulator. The ease of use in the case of the SPD Slingshot metamodel is based on a default parameter selection that is based on experimentation (see Section 7.2) and a lower-complexity model rather than a meta-learning approach. Still, the components of this meta-learning metamodel could in the future also be adopted for the SPD metamodel and further ease the use of machine learning. The proposed SPD metamodel could be a first step towards reaching a meta-learning metamodel for scaling policies.

The SPD metamodel can also easily be extended to support different models as there only needs to be one new type of model added and the model-specific behavior needs to be specified in Slingshot, for which more detail is provided in Section 6.2.

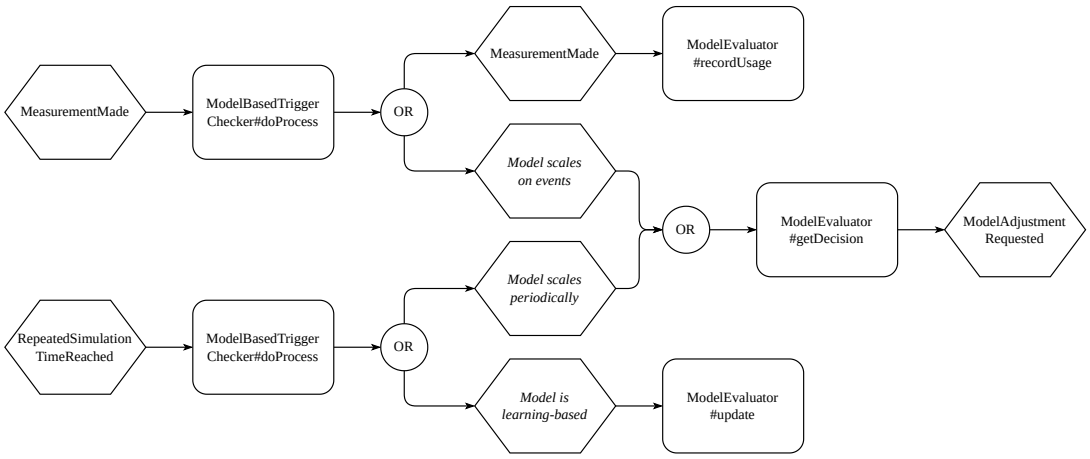


Figure 6.4: EPC depicting the event flow for the ModelBasedScalingPolicy implementation in Slingshot. Events in cursive are not Slingshot events but instead represent decision branches before method calls.

6.2 Slingshot implementation

In addition to the SPD metamodel extension described in Section 6.1, the model behaviors are also implemented in the Slingshot Simulator. An overview of the classes added for the implementation can be seen in Figure 6.3 The SPD interpretation process for a `ModelBasedScalingPoliy` will be described in Section 6.2.1. The runtime behavior and interaction between the newly added classes will be explained in Section 6.2.2.

6.2.1 Scaling Policy Interpretation

The Slingshot simulator needs to interpret the used scaling policy first in order to construct the objects that correspond to the scaling policy and are then used in the simulation. This is done in the `SpdInterpreter` class. For the new `ModelBasedScalingPolicy`, the `SpdInterpreter` creates a `ModelBasedTriggerChecker` which listens to both `RepeatedSimulationTimeReached` and `MeasurementMade` events. Each `ModelBasedTriggerChecker` contains a `ModelEvaluator` which itself contains the needed logic for accumulating and aggregating measurements and for deciding on scaling actions. The `ModelEvaluator` is created using the `ModelInterpreter`, which looks at the SPD policy and creates the model evaluator associated with the model that is contained in a scaling policy. Finally, the `SpdInterpreter` schedules a `RepeatedSimulationTimeReached` event that is based on the `interval` and `intervalOffset` attributes of the interpreted scaling policy.

A `ModelBasedTriggerChecker` instance is constructed for each `ModelBasedScalingPolicy` and coordinates how the received `MeasurementMade` and `RepeatedSimulationTimeReached` events should be forwarded to the `Model` instance that calculates the necessary adjustment.

6.2.2 Runtime behavior

The event flow of the new implementation can be seen in Figure 6.4. The `RepeatedSimulationTimeReachedRepeater` is a Slingshot behavior extension that listens to `RepeatedSimulationTimeReached` events and creates new `RepeatedSimulationTimeReached` events with the interval spacing specified in the event.

The `ModelBasedTriggerChecker` facilitates the scaling actions. Depending on whether a model is event-based or periodical, a scaling decision is made for each `MeasurementMade` or each `RepeatedSimulationTimeReached` event that is received by the `ModelBasedTriggerChecker`.

If a model is learning-based, the `update` method of the `ModelEvaluator` is first called. For Fuzzy Learning models, this method performs an iteration of the RL algorithm described in Section 5.2.1 until the point where the action values get updated. After the `update` method, the `getDecision` method is called for both non-learning-based and learning-based models. For the `RandomModelEvaluator`, a random number is generated and a scaling decision is made randomly based on the drawn random number and the probabilities obtained from the `RandomModel` contained in the corresponding scaling policy. For Fuzzy Learning models, the action that was calculated in the `update` method is simply returned in the `getDecision` method. The decision is then stored in the `adjustment` attribute of the scaling policy by the `ModelBasedTriggerChecker`. If an adjustment needs to be made, a `ModelAdjustmentRequested` event is returned, which ultimately leads to a scaling action being performed using a QVTo Transformator. The QVTo Transformator then obtains the calculated adjustment by reading the `adjustment` attribute of the model.

Before reaching the QVTo Transformator, the `ModelAdjustmentRequested` event needs to get through a `FilterChain` which can contain filters that enforce scaling constraints such as the thrashing constraint. These constraint filters have been modified to support the new model-based scaling policies.

For each `MeasurementMade` event received by the `ModelBasedTriggerChecker`, the measurement is also presented to the Model using the `recordUsage` method such that it can be recorded and aggregated if necessary.

As the `ModelBasedTriggerChecker` facilitates the event reception and forwards them to the model, the `ModelEvaluator` only needs the implementation of core model functionality, such as the initialization of stimulus aggregators or recorders and the decision-making process of the model. Thus, the implementation of a new model is rather easy and doesn't need a deep understanding of the Slingshot simulator.

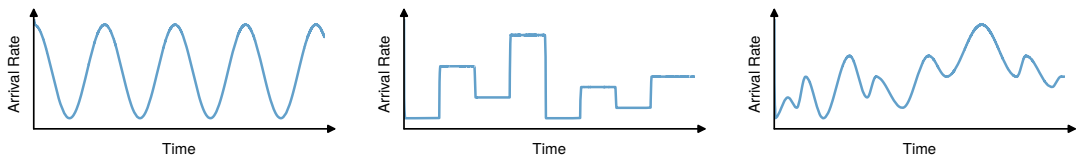
7 Evaluation

In this chapter, the evaluation is performed. First, the design of the study is explained. Then, the results are shown and discussed. Possible threats to validity are also mentioned. This chapter answers the research questions **RQ3.1**: How do the different approaches compare to each other when using the Slingshot simulator? and **RQ3.2**: Can the claims made in the accompanying papers be verified using the Slingshot simulator? The evaluation dataset containing the results of the evaluation runs as well as the used models is provided on Zenodo [Ber24].

7.1 Study Design

For the evaluation, it was chosen to replicate the evaluation design of the study from which the implemented scaling approaches were taken. This means that the results can be compared to the original study results and it can be verified that the scaling approach behaves similarly inside the slingshot simulator and in the real world. The evaluation thus entails first designing a Palladio Model and SPD that replicates the system architecture of the original evaluation. Then, the used workload patterns also need to be replicated using Descartes Load Intensity Models [KHK+17].

In the paper corresponding to the implemented approach [APJE17], different evaluations using both synthetic and real-world workload patterns were performed. As this evaluation does not only compare the implementation to the results of the original paper but also contains some additional evaluations, only the synthetic workload patterns will be discussed. The replicated workload patterns are the Predictable Bursting pattern that models a sinusoidal load shown in Figure 7.1a, the On-Off pattern shown in Figure 7.1b and the Variations pattern shown in Figure 7.1c. The workload patterns peak at a rate of 100 requests per second for the Predictable Bursting and Variations patterns or 90 requests for the On-Off pattern and have a minimum rate of 10 requests per second.



(a) Predictable Bursting **(b)** On-Off **(c)** Variations

Figure 7.1: The different workload patterns

7.1.1 Palladio Modeling

The architecture of the evaluation performed by Arabnejad et al. is described as a web server that is tested using an HTTP load generation tool and which can be scaled between one and five VMs [APJE17]. This setup is recreated as a PCM instance. The Palladio model used for the evaluation consists of a web server and a load balancer. The requests go through the load balancer which distributes the requests equally to the different running instances of the web server.

A difficulty in modeling the system was in the behavior a Slingshot model has when applying varying loads when compared to the real world. As can be seen in the original study, the response time of a system generally increases when the load increases, even when a system has spare capacities to handle additional requests [APJE17]. However, when naively modeling a Slingshot model using an arrival rate, the behavior is different: The response time does not increase until the load is too high for the system, at which point the response time increases linearly if the load stays the same. This can be seen in Figure 7.2. After the request arrival rate reaches 100, the system is not able to keep up with the incoming requests and the response time increases linearly. When the request arrival rate decreases, the response time decreases too after the queued requests have been processed.

In order to more accurately represent the real world in Slingshot, randomness can be used to ensure that the requests do not go through the system in a perfectly spaced way. It was first tried to introduce some randomness by introducing a delay in the modeled network. However, the response time behavior for a network with delays shown in Figure 7.3 was unexpected. Even when only introducing a sufficiently large constant network delay, the response times increases linearly with time instead of only being higher by the amount of the delay. Therefore, the delay was instead placed before the user request in the usage model. Like this, the behavior of the simulated system shown in Figure 7.4 was similar to the real-world behavior described by Arabnejad et al. of having a higher response time when the system load increases, even when the system is not fully saturated by the load [APJE17].

7.1.2 Evaluation

The evaluation mainly focuses on two metrics to compare the different scaling policies. The average number of containers shows how many containers are allocated on average over the whole duration of the workload. A lower number of average containers is preferable as this reduces the running cost of a system, as the containers are homogeneous and thus assumed to be priced the same. Additionally, the adherence to a SLO was tested. This SLO was determined by running an evaluation using the different workloads for both five and one statically assigned containers without any scaling. The SLO was then chosen such that it is achievable when using five containers, but not when using one container. For this, the SLO was determined to be that the 90th percentile of response times should not be larger than 20ms. To get a usable SLO violation number, the results of each experiment were divided into one-minute intervals and the adherence to the SLO was checked for each interval and the ratio of SLO violations was then determined by using the following formula:

$$\text{SLO Violation \%} = \frac{\text{One Minute Intervals where 90-th percentile response time} > 0.02}{\text{One Minute Intervals in total}} \tag{7.1}$$

When only using one container, this yields worst-case SLO violations of 52%, 65%, and 72% for the On-Off, Predictable Bursting, and Variations workload patterns respectively.

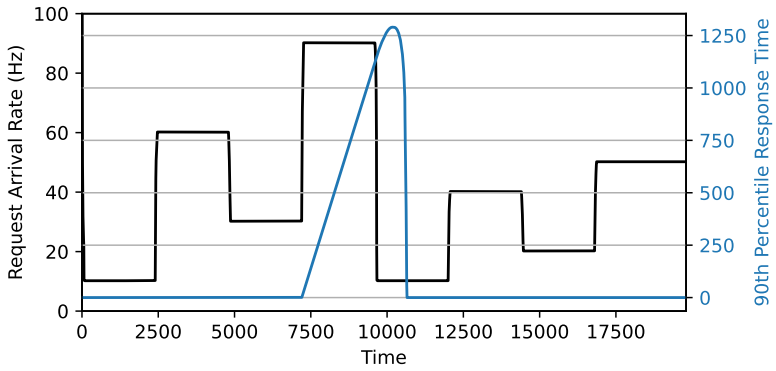


Figure 7.2: Response time behavior for system that is overloaded

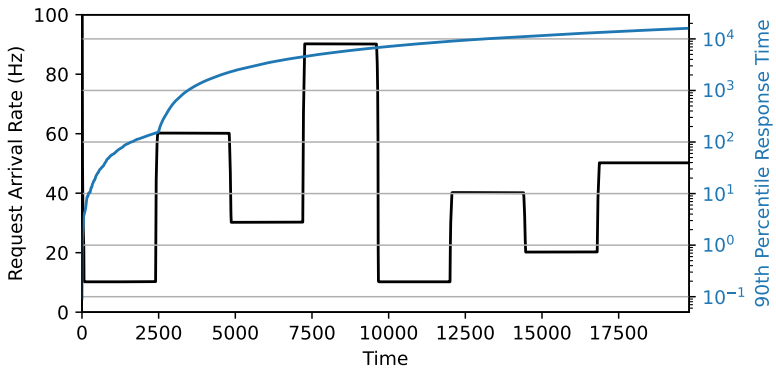


Figure 7.3: Unexpected behavior of response times with randomized delay at network level

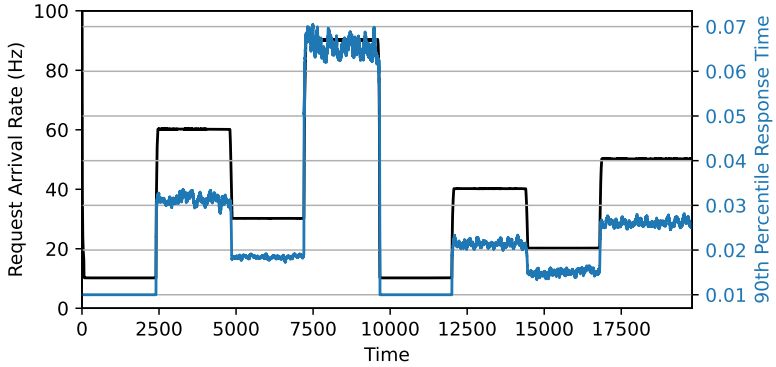


Figure 7.4: Response times with randomized delay at user arrival

7.2 Results

First, different configurations of the proposed model are compared. The behavior of the model is also shown, focusing especially on the initial behavior in the exploration phase. Then, the Fuzzy SARSA model is compared with the Fuzzy Q-Learning model and different reactive scaling methods. The given target for the Fuzzy Learning models was a maximum 90th percentile response time of 20ms. The graphs each show the mean value for the different scaling policies and the different load patterns.

7.2.1 Model configuration

The comparison of different Fuzzy SARSA model configurations focuses on three configuration parameters: Whether adding a reward for scaling actions improves the performance and how different SLO goal percentiles and different Q-Value initialization strategies affect the model performance. Each evaluation focuses on one parameter. The SLO goal of the fuzzy learning model was set to the 90th percentile, while the other model parameters were left at the default value as specified in the metamodel. The goal of these experiments is to find the best configuration for the model.

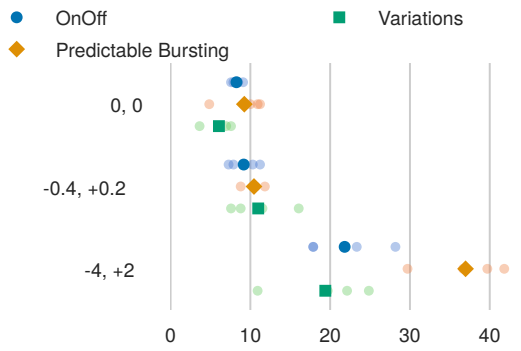
Scaling reward

As the model tends to do too many scaling actions, an experiment was performed where the reward function was modified with added rewards r_a for scaling actions as described in Section 5.2. r_a should discourage scaling actions in general, as the action reward for scaling up and then down the same amount is negative. It should also help keep the container count as low as possible by giving a penalty for scaling out and a small reward for scaling in.

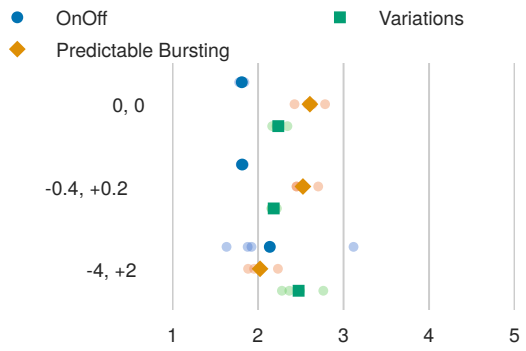
However, the results in Figure 7.5 indicate that the action reward discourages scaling too much and at the wrong time, leading to a worsened SLO adherence. For a high scaling reward and penalty, the percentage of SLO violations increases significantly to up to 40% while the container count is reduced. For the moderate scaling penalty of $r_{\text{out}} = 0.4$ for scaling out and a reward of $r_{\text{in}} = 0.2$ for scaling in, the number of scaling actions only decreases slightly while the performance worsens slightly. Due to these results, no scaling action reward is used for the other evaluations.

SLO goals

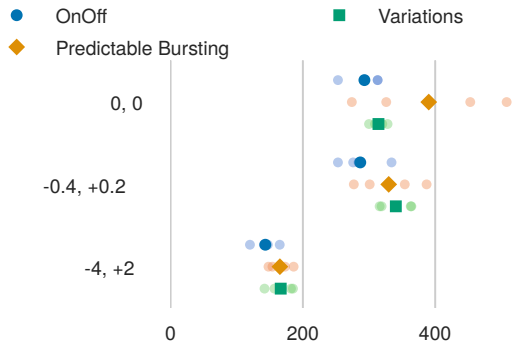
It was also evaluated whether having a stricter SLO goal for the reward function of the fuzzy learning models might improve their performance. By giving the model a 95-th percentile response time target, the adherence to the actual 90th percentile response time SLO can be improved. For the Predictable Bursting workload, the share of timeframes where the SLO was violated decreased from 9.2% to 2.3% when using a stricter SLO for the model than necessary, while the average used container count increased moderately from 2.6 to 3.2. To ensure strict adherence to an SLO, it might thus be preferable to give the learning model a stricter SLO.



(a) SLO Violations in %

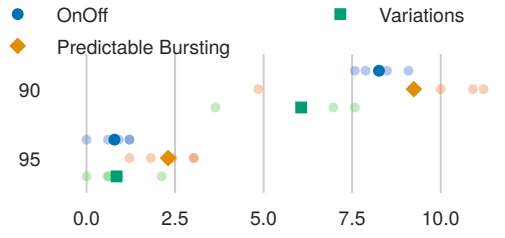


(b) Average container count

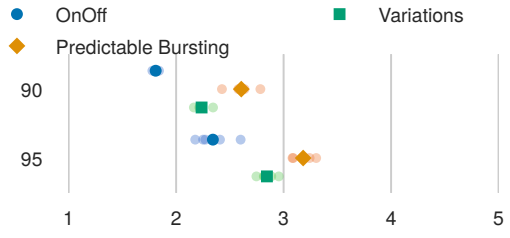


(c) Number of scaling actions

Figure 7.5: Comparison of Model Performance for different scaling rewards

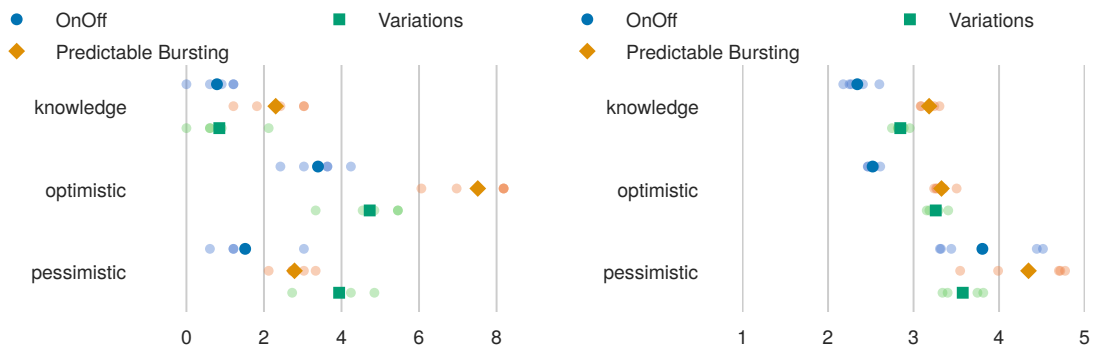


(a) SLO Violations in %



(b) Average container count

Figure 7.6: Comparison of Model Performance for different SLO goal percentiles



(a) SLO Violations in %

(b) Average container count

Figure 7.7: Comparison of Model Performance for different initialization strategies

Initialization strategies

Finally, the different methods of initialization discussed in Section 5.2 are evaluated. As can be seen in Figure 7.7, the optimistic initialization has on average the worst performance in terms of violations for all tested scenarios while the SLO violation share is lowest for the pessimistic initialization. For the Predictable Bursting scenario, the model using an optimistic initialization violates the SLO in 7.5% of the timeframes, while the pessimistic and knowledge initializations only violate the SLO for 2.8% and 2.3% of the timeframes respectively. However, in the case of the pessimistic initialization strategy, this comes at the cost of a much higher average container count of 4.3 compared to the 3.2 and 3.3 containers used by the knowledge and optimistic approaches. The results are similar for the Variatons and On-Off patterns. As the knowledge-based initialization has a slightly lower share of SLO violations compared to the pessimistic initialization for most of the tested scenarios while also using fewer containers, it is chosen as the preferred initialization method.

7.2.2 Initial Performance

An inherent downside of Reinforcement Learning models is that they start with limited or no knowledge and first try to explore the state space. This leads to worse initial performance. Figure 7.8 shows the allocated number of containers, the arrival rate, and the resulting response time for an evaluation using the Fuzzy SARSA model for scaling and the Variations workload. The response time represents the 90th percentile response time for the last minute. It can be seen that especially in the beginning, the model makes many scaling decisions and sometimes uses far too many containers, e.g., using five containers when one would be sufficient. This downside partially extends into the later exploitation phase as well, where random ε-decisions are still taken with a probability of 5%. This leads to the container count curve being less smooth than it could ideally be.

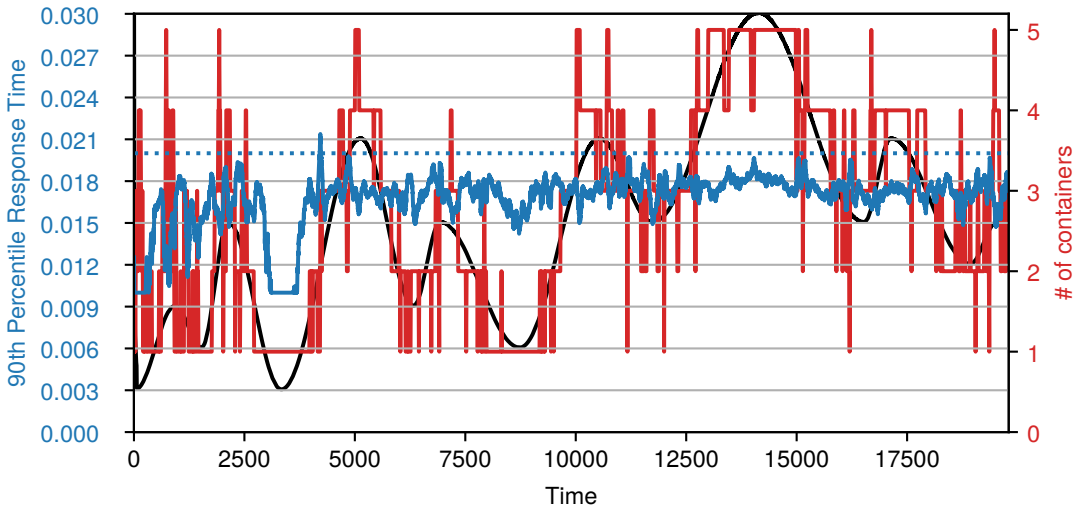
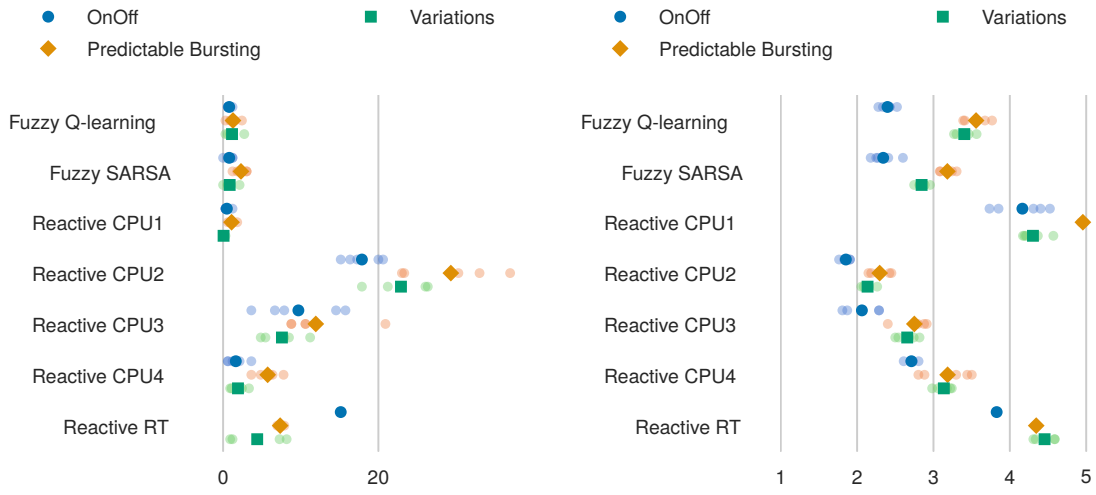


Figure 7.8: Example run of a Fuzzy SARSA model



(a) SLO Violations in %

(b) Average container count

Figure 7.9: Comparison of Model Performance to Reactive Scaling Policies

7.2.3 Comparison between Fuzzy learning policies and simple reactive scaling policies

Additionally, the model using the parameters that performed best in the previous section, i.e. a knowledge-based initialization and a 95th percentile SLO target, is compared with different reactive scaling policies in Figure 7.9. The compared scaling policies are the following:

- CPU1: Scaling in when the CPU load is at 30%, scaling out for a CPU load of 80%
- CPU2: Scaling in when the CPU load is at 50%, scaling out for a CPU load of 80%
- CPU3: Scaling in when the CPU load is at 45%, scaling out for a CPU load of 80%
- CPU4: Scaling in when the CPU load is at 45%, scaling out for a CPU load of 75%
- RT: Scaling out when the response time is at 18ms, scaling in at 10ms while not allowing any thrashing, i.e. scaling actions in opposite directions, for 10s.

These policies were created iteratively. First, CPU1 and RT were conceived. After they were evaluated, new policies addressing their weak points were created. For example, as the CPU1 policy used too many containers, the threshold for scaling in was increased for the CPU2 policy. The response-time-based scaling policy additionally has a limit on thrashing, as response-time policies without any thrashing limit perform an excessive amount of scaling actions.

The reactive scaling policy CPU4 that scaled based on CPU load and scaled in at 45% load while scaling out at 75% load yielded the best load out of the tested reactive policies. It used between 2.7 and 3.2 containers on average compared to the 2.3-3.6 containers used by the fuzzy learning models. When comparing the ratio of SLO violations, the reactive scaling policy violated the SLO for 1.6-5.8% of the timeframes while the Fuzzy Learning models only violated the SLO for 0.8-2.3% of the timeframes. However, despite the lower SLO performance while using more containers, the scaling behavior of the reactive model still has some advantages. The model only performed between 110 and 188 scaling actions on average while the fuzzy learning models performed more than double of these at between 269 and 445 scaling actions.

When comparing the performance in the first 1000s, the fuzzy learning models have the highest rate of SLO violations for the Predictable Bursting pattern at 18 to 59%, while the CPU2 scaling policy has a SLO violation rate of 11 to 35%. This clearly shows the disadvantage of learning-based models within their learning phase compared to the reactive scaling policies that do not have a learning phase.

Fuzzy SARSA and Fuzzy Q-learning exhibit similar behavior for the On-Off workload, while Fuzzy SARSA performs better with a higher SLO adherence and lower resource usage for the Predictable Bursting and Variations workload patterns.

7.2.4 Comparison to the results of the original paper

When comparing the results of the Slingshot implementation of Fuzzy SARSA and Fuzzy Q-Learning to the results presented by Arabnejad et al. [APJE17, p. 8], the observed behavior seems to differ. The Slingshot implementations of the algorithms have a less variable response time, mostly

resting between 10 and 30 ms, while the original models seem to have response times in the range of 5 to 60 ms. This is most likely caused by the different behavior of the Slingshot simulated model and the real-world performance.

While Arabnejad et al. describe Fuzzy SARSA as having a tendency to obtain more VMs [APJE17, p. 9], as shown in Figure 7.9, for the Slingshot implementations, the Fuzzy SARSA approach uses fewer containers than Fuzzy Q-Learning. This difference could be caused by the different reward function that is used in Slingshot, as the reward function was not disclosed in the original paper.

As shown in Section 7.2.1, the finding that good initial knowledge leads to improved model performance [APJE17, p. 9] can be replicated in Slingshot. Another finding that can be replicated in Slingshot is the similar performance of the Fuzzy Q-Learning and Fuzzy SARSA approaches for the On-Off workload.

7.3 Discussion

As shown in the previous section, especially the Fuzzy SARSA model manages to exceed the performance of simple reactive scaling models. Thus, the addition of this new class of scaling policies in the SPD metamodel and the Slingshot simulator can make them more versatile and enable better scaling behavior. However, this comes with a few downsides. As described in Section 7.2.2, the reinforcement learning model first needs a lengthy exploration phase to find a good policy. Thus, evaluations performed with a reinforcement learning model need to be longer compared to evaluations performed with non-learning-based models. Within the context of this evaluation, the simulations all encompassed 19800 simulated seconds, as this was the timeframe used in the original evaluations. This led to quite lengthy running times per simulation. Each simulation took around one hour to complete on a modern desktop computer.

Creating reactive scaling policies that more closely reflect the performance of the fuzzy learning model-based scaling policies might be possible, however, this would necessarily be connected to a higher effort and some trial and error. The model-based scaling policies provide an easy-to-use model where this trial and error load is not incurred on the modeler but instead performed by the model itself. Like this, they can also lower the barrier of entry to achieving good scalability behavior and make the application of scalability to an architectural model more attractive overall.

The comparison of the Slingshot versions of Fuzzy Q-learning and Fuzzy SARSA to the results of Arabnejad et al. shows that the exact replication of real-world results within Slingshot is difficult when not all model and environment parameters are known. However, the broad behavioral trends such as the benefit of good initialization values can be observed both within Slingshot and in the real-world evaluation.

The evaluation also shows that scaling policies that scale based on the response time are currently not performing well using the Slingshot approach. This is caused by the response time variations that are caused by the randomization used in the PCM model. Extending the SPD metamodel to support the aggregation of response times using a percentile or average over some timespan and using this aggregated value as an input for a scaling policy trigger could improve the performance for this case.

7.4 Threats to Validity

A threat to validity is the randomness that is inherent to both the PCM instance used for the simulation and the reinforcement learning method in general. This randomness makes it harder to compare the results of the different models. This was counteracted by repeating the evaluation for each model at least four times. It was also determined that for static models, the variance is pretty small - thus, the randomization of the Palladio model should not influence the results. For example, for one static container, the percentage of timeframes where the SLO was violated was 64.74% with a standard deviation of 0.17%.

Another threat to validity is that the learning-based model was only compared to a small number of relatively simple reactive scaling policies. However, simple scaling policies that are based on one metric such as the CPU load or the response time are commonly used in practice [LML14, p. 566]. This threat was also counteracted by including different reactive scaling policies that were iteratively constructed by looking at the results of previous scaling policies and adjusting the policy based on them into the evaluation such that reactive scaling policies should at least be representative of scaling policies created by a novice modeler.

A different threat to validity is the possible presence of bugs in the fuzzy learning slingshot implementation that result in a different behavior of the RL models than expected. This is counteracted by sanity-checking all the results, so the presence of major bugs that affect the evaluation should be unlikely.

8 Conclusion

In this chapter, a summary of the thesis is given. Additionally, the benefits and limitations of the results of the thesis are discussed. Finally, the learned lessons and potential future work are presented.

8.1 Summary

In this thesis, the SPD metamodel was extended to support learning-based scaling policies. Using a rapid literature review, the Fuzzy Q-learning and Fuzzy SARSA methods proposed by Arabnejad et al. [APJE17] were chosen for the implementation. In order to facilitate the use of these new scaling policies, several evaluations were performed to find a good set of default parameters for these models. The new learning-based scaling policies performed better than simple reactive scaling policies and can cut down the work required to find a good scaling strategy. However, learning-based scaling policies have the fundamental downside of being unsuitable for short evaluations, as the models first need to have a learning phase before offering performance that is comparable to reactive scaling policies.

8.2 Benefits

Software architects performing performance evaluations using the Slingshot simulator can now use model-based scaling policies to understand their impact on the application scalability behavior for their architecture and to decide whether model-based scaling policies can be of an advantage to them.

The result can also be of benefit to developers of model-based scaling approaches, as the SPD extension and Slingshot simulator implementation can be used to easily integrate new models into the simulator and to then evaluate and compare the performance of these models. Compared to an evaluation on real cloud systems, this approach is faster and allows easier repeatability and comparability as the SPD model can easily be shared, and evaluations can be repeated using the same model, possibly yielding more consistent results.

8.3 Limitations

While the introduced model-based scaling policies can offer improved performance compared to reactive scaling policies, they also come with some drawbacks. In comparison to reactive scaling policies, they cannot be as easily understood by a modeler, as the adjustments are based on the

dynamically learned behavior. Additionally, the RL-based scaling policies presented in the thesis all have an initial learning phase in which a reactive policy typically outperforms these approaches. An inherent limitation of these scaling policies is that for exploration, the model sometimes performs random actions that might not lead to the desired result and could thus result in momentarily degraded performance and violated SLOs.

8.4 Lessons Learned

During the writing of the thesis, multiple lessons have been learned. First, the importance of iteratively designing a metamodel and trying out different model structures before deciding on a result was made clear. This helped with coming up with a good SPD metamodel extension.

Another learned lesson was to always look out for patents when implementing methods described in research papers, even if the papers themselves do not mention them.

Lastly, the difficulty of reliably evaluating the outcomes of an RL model that is not deterministic was also learned.

8.5 Future Work

There are several directions in which new work can build on the results of this thesis. First, the SPD metamodel and the Slingshot simulator could be extended to further move towards the goal of a meta-learning meta-model as proposed by Hartmann et al. [HMS+19] and discussed in Section 6.1.2. For this, a new model construct that signifies an arbitrary model that is chosen by some learning component would need to be introduced and the Slingshot simulator would need to be extended with such a learning component.

Another direction in which the work could be extended is the implementation of new models and the comparison of them. In the existing literature on learning-based cloud autoscaling, the proposed models are often evaluated using custom benchmarks and are thus not easily comparable. By adding different learning-based models to the SPD metamodel and implementing them in the Slingshot simulator, a new benchmark for models could be established. Additionally, to be able to perform a more varied evaluation, a set of systems would need to be modeled using the PCM, and further workload traces would need to be specified using Descartes Load Intensity Model [KHK+17], as the currently used traces are all purely synthetic.

An extension of the model-based scaling policies that provides an output of the learned model that can easily be reused, e.g., as knowledge initialization for a learning-based model or even as a non-learning model, would also be a possible future work. Like this, the Slingshot simulator could also be used to learn optimal scaling behaviors for a model in a safe environment and then transfer the model learnings to the real world.

Bibliography

[AHZ17] Y. Arian, S. Horovitz, M. Zheng. “Q-Learning Based Resource Scheduling Method and Device”. WO2017201662A1. Nov. 2017 (cit. on p. 20).

[AI24] S. N. Agos Jawaddi, A. Ismail. “Integrating OpenAI Gym and CloudSim Plus: A Simulation Environment for DRL Agent Training in Energy-Driven Cloud Scaling”. In: *Simulation Modelling Practice and Theory* 130 (Jan. 2024), p. 102858. ISSN: 1569190X. DOI: 10.1016/j.simpat.2023.102858. (Visited on 05/14/2024) (cit. on p. 7).

[AKTE12] A. Ali-Eldin, M. Kihl, J. Tordsson, E. Elmroth. “Efficient Provisioning of Bursty Scientific Workloads on the Cloud Using Adaptive Elasticity Control”. In: *Proceedings of the 3rd Workshop on Scientific Cloud Computing*. Delft The Netherlands: ACM, June 2012, pp. 31–40. ISBN: 978-1-4503-1340-7. DOI: 10.1145/2287036.2287044. (Visited on 06/11/2024) (cit. on pp. 12–18, 20, 58).

[Ama] Amazon. *Predictive Scaling for Amazon EC2 Auto Scaling - Amazon EC2 Auto Scaling*. https://docs.aws.amazon.com/autoscaling/ec2/userguide/ec2-auto-scaling-predictive-scaling.html. (Visited on 12/08/2024) (cit. on p. 1).

[APJE17] H. Arabnejad, C. Pahl, P. Jamshidi, G. Estrada. *A Comparison of Reinforcement Learning Techniques for Fuzzy Cloud Auto-Scaling*. May 2017. arXiv: 1705.07114 [cs]. (Visited on 06/12/2024) (cit. on pp. 12–20, 24, 35, 36, 42, 43, 45, 58).

[ATE12] A. Ali-Eldin, J. Tordsson, E. Elmroth. “An Adaptive Hybrid Elasticity Controller for Cloud Infrastructures”. In: *2012 IEEE Network Operations and Management Symposium*. Maui, HI: IEEE, Apr. 2012, pp. 204–212. ISBN: 978-1-4673-0269-2 978-1-4673-0267-8 978-1-4673-0268-5. DOI: 10.1109/NOMS.2012.6211900. (Visited on 06/18/2024) (cit. on pp. 12–19, 58).

[BBD+16] A. Basiri, N. Behnam, R. De Rooij, L. Hochstein, L. Kosewski, J. Reynolds, C. Rosenthal. “Chaos Engineering”. In: *IEEE Software* 33.3 (May 2016), pp. 35–41. ISSN: 0740-7459, 1937-4194. DOI: 10.1109/MS.2016.60. (Visited on 10/05/2024) (cit. on p. 30).

[BD19] J. V. Bibal Benifa, D. Dejey. “RLPAS: Reinforcement Learning-based Proactive Auto-Scaler for Resource Provisioning in Cloud Environment”. In: *Mobile Networks and Applications* 24.4 (Aug. 2019), pp. 1348–1363. ISSN: 1572-8153. DOI: 10.1007/s11036-018-0996-0. (Visited on 06/12/2024) (cit. on pp. 12–19, 58).

[Bec10] S. Becker. “The Palladio Component Model”. In: *Proceedings of the First Joint WOSP/SIPEW International Conference on Performance Engineering*. San Jose California USA: ACM, Jan. 2010, pp. 257–258. ISBN: 978-1-60558-563-5. DOI: 10.1145/1712605.1712651. (Visited on 04/30/2024) (cit. on pp. 1, 3).

[Ber24] J. Berberich. *Evaluation of Fuzzy Learning-Based Scaling Policies for the Slingshot Simulator*. Dec. 2024. DOI: 10.5281/zenodo.14276724 (cit. on p. 35).

[BGS+09] P. Bodík, R. Griffith, C. Sutton, A. Fox, M. I. Jordan, D. A. Patterson. "Statistical Machine Learning Makes Automatic Control Practical for Internet Datacenters." In: *HotCloud* 9 (2009), pp. 12–12 (cit. on pp. 12–18, 60).

[BHD13] E. Barrett, E. Howley, J. Duggan. "Applying Reinforcement Learning towards Automating Resource Allocation and Application Scalability in the Cloud". In: *Concurrency and Computation: Practice and Experience* 25.12 (Aug. 2013), pp. 1656–1674. ISSN: 1532-0626, 1532-0634. DOI: 10.1002/cpe.2864. (Visited on 06/12/2024) (cit. on pp. 12–18, 58).

[BKR07] S. Becker, H. Koziolek, R. Reussner. "Model-Based Performance Prediction with the Palladio Component Model". In: *Proceedings of the 6th International Workshop on Software and Performance*. Buenes Aires Argentina: ACM, Feb. 2007, pp. 54–65. ISBN: 978-1-59593-297-6. DOI: 10.1145/1216993.1217006. (Visited on 04/30/2024) (cit. on pp. iii, v, 1, 3).

[BRC09] R. Buyya, R. Ranjan, R. N. Calheiros. "Modeling and Simulation of Scalable Cloud Computing Environments and the CloudSim Toolkit: Challenges and Opportunities". In: *2009 International Conference on High Performance Computing & Simulation*. Leipzig, Germany: IEEE, June 2009, pp. 1–11. ISBN: 978-1-4244-4906-4. DOI: 10.1109/HPCSIM.2009.5192685. (Visited on 04/30/2024) (cit. on p. 7).

[BVU+10] D. A. Bacigalupo, J. Van Hemert, A. Usmani, D. N. Dillenberger, G. B. Wills, S. A. Jarvis. "Resource Management of Enterprise Cloud Systems Using Layered Queuing and Historical Performance Models". In: *2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW)*. Atlanta, GA, USA: IEEE, Apr. 2010, pp. 1–8. ISBN: 978-1-4244-6533-0. DOI: 10.1109/IPDPSW.2010.5470782. (Visited on 06/18/2024) (cit. on pp. 12–18, 59).

[CBY19] T. Chen, R. Bahsoon, X. Yao. "A Survey and Taxonomy of Self-Aware and Self-Adaptive Cloud Autoscaling Systems". In: *ACM Computing Surveys* 51.3 (May 2019), pp. 1–40. ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3190507. (Visited on 05/08/2024) (cit. on pp. 7, 9, 10).

[CCG13] H. Chihi, W. Chainbi, K. Ghedira. "An Energy-Efficient Self-Provisioning Approach for Cloud Resources Management". In: *ACM SIGOPS Operating Systems Review* 47.3 (Nov. 2013), pp. 2–9. ISSN: 0163-5980. DOI: 10.1145/2553070.2553072. (Visited on 06/17/2024) (cit. on pp. 12–18, 58).

[CDM10] E. Caron, F. Desprez, A. Muresan. "Forecasting for Grid and Cloud Computing On-Demand Resources Based on Pattern Matching". In: *2010 IEEE Second International Conference on Cloud Computing Technology and Science*. Nov. 2010, pp. 456–463. DOI: 10.1109/CloudCom.2010.65. (Visited on 06/17/2024) (cit. on pp. 12–18, 59).

[CDM11] E. Caron, F. Desprez, A. Muresan. "Pattern Matching Based Forecast of Non-periodic Repetitive Behavior for Cloud Clients". In: *Journal of Grid Computing* 9.1 (Mar. 2011), pp. 49–64. ISSN: 1572-9184. DOI: 10.1007/s10723-010-9178-4. (Visited on 06/20/2024) (cit. on pp. 12–18, 59).

[CGS03] A. Chandra, W. Gong, P. Shenoy. "Dynamic Resource Allocation for Shared Data Centers Using Online Measurements". In: (2003) (cit. on pp. 12–18, 60).

[CH11] R. C. Chiang, H. H. Huang. “TRACON: Interference-Aware Scheduling for Data-Intensive Applications in Virtualized Environments”. In: (2011) (cit. on pp. 12–18, 59).

[CHL+08] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, F. Zhao. “Energy-Aware Server Provisioning and Load Dispatching for Connection-Intensive Internet Services”. In: (2008) (cit. on pp. 12–18, 60).

[CPS20] B. Cartaxo, G. Pinto, S. Soares. “Rapid Reviews in Software Engineering”. In: *Contemporary Empirical Methods in Software Engineering*. Ed. by M. Felderer, G. H. Travassos. Cham: Springer International Publishing, 2020, pp. 357–384. ISBN: 978-3-030-32489-6. DOI: 10.1007/978-3-030-32489-6_13 (cit. on p. 9).

[CS13] E. Casalicchio, L. Silvestri. “Autonomic Management of Cloud-Based Systems: The Service Provider Perspective”. In: *Computer and Information Sciences III*. Springer, London, 2013, pp. 39–47. ISBN: 978-1-4471-4594-3. DOI: 10.1007/978-1-4471-4594-3_5. (Visited on 06/18/2024) (cit. on pp. 12–18, 58).

[DCS+19] J. Ding, R. Cao, I. Saravanan, N. Morris, C. Stewart. “Characterizing Service Level Objectives for Cloud Services: Realities and Myths”. In: *2019 IEEE International Conference on Autonomic Computing (ICAC)*. Umea, Sweden: IEEE, June 2019, pp. 200–206. ISBN: 978-1-72812-411-7. DOI: 10.1109/ICAC.2019.00032. (Visited on 11/20/2024) (cit. on p. 23).

[DGVV12] S. Dutta, S. Gera, A. Verma, B. Viswanathan. “SmartScale: Automatic Application Scaling in Enterprise Clouds”. In: *2012 IEEE Fifth International Conference on Cloud Computing*. June 2012, pp. 221–228. DOI: 10.1109/CLOUD.2012.12. (Visited on 06/20/2024) (cit. on pp. 12–18, 59).

[DKM+11] X. Dutreilh, S. Kirgizov, O. Melekhova, J. Malenfant, N. Rivierre. “Using Reinforcement Learning for Autonomic Resource Allocation in Clouds: Towards a Fully Automated Workflow”. In: (2011) (cit. on pp. 12–18, 59).

[DMM+10] X. Dutreilh, A. Moreau, J. Malenfant, N. Rivierre, I. Truck. “From Data Center Resource Allocation to Control Theory and Back”. In: *2010 IEEE 3rd International Conference on Cloud Computing*. Miami, FL, USA: IEEE, July 2010, pp. 410–417. ISBN: 978-1-4244-8207-8. DOI: 10.1109/CLOUD.2010.55. (Visited on 06/18/2024) (cit. on pp. 12–19, 59).

[DS18] N. Dezhabad, S. Sharifian. “Learning-Based Dynamic Scalable Load-Balanced Firewall as a Service in Network Function-Virtualized Cloud Computing Environments”. In: *The Journal of Supercomputing* 74.7 (July 2018), pp. 3329–3358. ISSN: 1573-0484. DOI: 10.1007/s11227-018-2387-5. (Visited on 06/12/2024) (cit. on pp. 12–19, 58).

[FLWC12] W. Fang, Z. Lu, J. Wu, Z. Cao. “RPPS: A Novel Resource Prediction and Provisioning Scheme in Cloud Data Center”. In: *2012 IEEE Ninth International Conference on Services Computing*. June 2012, pp. 609–616. DOI: 10.1109/SCC.2012.47. (Visited on 06/20/2024) (cit. on pp. 12–18, 59).

[GJ97] P. Glorennec, L. Jouffe. “Fuzzy Q-learning”. In: *6th International Fuzzy Systems Conference*. Vol. 2. Barcelona, Spain: IEEE, 1997, pp. 659–662. ISBN: 978-0-7803-3796-1. DOI: 10.1109/FUZZY.1997.622790. (Visited on 11/25/2024) (cit. on p. 24).

[GJP18] M. Ghobaei-Arani, S. Jabbehdari, M. A. Pourmina. “An Autonomic Resource Provisioning Approach for Service-Based Cloud Applications: A Hybrid Approach”. In: *Future Generation Computer Systems* 78 (Jan. 2018), pp. 191–210. ISSN: 0167739X. DOI: 10.1016/j.future.2017.02.022. (Visited on 06/12/2024) (cit. on pp. 12–18, 58).

[GMP+21] Y. Garí, D. A. Monge, E. Pacini, C. Mateos, C. García Garino. “Reinforcement Learning-Based Application Autoscaling in the Cloud: A Survey”. In: *Engineering Applications of Artificial Intelligence* 102 (June 2021), p. 104288. ISSN: 09521976. DOI: 10.1016/j.engappai.2021.104288. (Visited on 05/08/2024) (cit. on pp. 1, 7, 9, 10, 19).

[GT12] A. Gambi, G. Toffetti. “Modeling Cloud Performance with Kriging”. In: *2012 34th International Conference on Software Engineering (ICSE)*. Zurich: IEEE, June 2012, pp. 1439–1440. ISBN: 978-1-4673-1066-6 978-1-4673-1067-3. DOI: 10.1109/ICSE.2012.6227075. (Visited on 06/18/2024) (cit. on pp. 12–18, 58).

[HA18] S. Horovitz, Y. Arian. “Efficient Cloud Auto-Scaling with SLA Objective Using Q-Learning”. In: *2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud)*. Aug. 2018, pp. 85–92. DOI: 10.1109/FiCloud.2018.00020. (Visited on 06/12/2024) (cit. on pp. 12–20, 23, 58).

[HAZ18a] S. Horovitz, Y. Arian, M. Zheng. “Action Space Reduction for Cloud Auto-Scaling Applications”. WO2018127297A1. July 2018 (cit. on p. 20).

[HAZ18b] S. Horovitz, Y. Arian, M. Zheng. “Method and Device for Adjusting State Space Boundary in Q-Learning”. WO2018098797A1. June 2018 (cit. on p. 20).

[HLY12] J. Huang, C. Li, J. Yu. “Resource Prediction Based on Double Exponential Smoothing in Cloud Computing”. In: *2012 2nd International Conference on Consumer Electronics, Communications and Networks (CECNet)*. Yichang, China: IEEE, Apr. 2012, pp. 2056–2060. ISBN: 978-1-4577-1415-3 978-1-4577-1414-6 978-1-4577-1413-9. DOI: 10.1109/CECNet.2012.6201461. (Visited on 06/11/2024) (cit. on pp. 12–18, 59).

[HMS+19] T. Hartmann, A. Moawad, C. Schockaert, F. Fouquet, Y. Le Traon. “Meta-Modelling Meta-Learning”. In: *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS)*. Vol. 31. Munich, Germany: IEEE, Sept. 2019, pp. 300–305. DOI: 10.1109/models.2019.00014. (Visited on 09/17/2024) (cit. on pp. 31, 32, 46).

[IDCJ11] W. Iqbal, M. N. Dailey, D. Carrera, P. Janecek. “Adaptive Resource Provisioning for Read Intensive Multi-Tier Applications in the Cloud”. In: *Future Generation Computer Systems* 27.6 (June 2011), pp. 871–879. ISSN: 0167-739X. DOI: 10.1016/j.future.2010.10.016. (Visited on 06/20/2024) (cit. on pp. 12–18, 59).

[IKLL12] S. Islam, J. Keung, K. Lee, A. Liu. “Empirical Prediction Models for Adaptive Resource Provisioning in the Cloud”. In: *Future Generation Computer Systems* 28.1 (Jan. 2012), pp. 155–162. ISSN: 0167-739X. DOI: 10.1016/j.future.2011.05.027. (Visited on 06/20/2024) (cit. on pp. 12–18, 58).

[JPLC13] Y. Jiang, C.-S. Perng, T. Li, R. N. Chang. “Cloud Analytics for Capacity Planning and Instant VM Provisioning”. In: *IEEE Transactions on Network and Service Management* 10.3 (Sept. 2013), pp. 312–325. ISSN: 1932-4537. DOI: 10.1109/TNSM.2013.051913.120278. (Visited on 06/17/2024) (cit. on pp. 12–18, 58).

[JSP+15] P. Jamshidi, A. M. Sharifloo, C. Pahl, A. Metzger, G. Estrada. “Self-Learning Cloud Controllers: Fuzzy Q-Learning for Knowledge Evolution”. In: *2015 International Conference on Cloud and Autonomic Computing (ICCAC)*. Boston, MA, USA: IEEE, Sept. 2015, pp. 208–211. ISBN: 978-1-4673-9566-3. DOI: 10.1109/ICCAC.2015.35. (Visited on 10/15/2024) (cit. on p. 24).

[JSP+16] P. Jamshidi, A. Sharifloo, C. Pahl, H. Arabnejad, A. Metzger, G. Estrada. “Fuzzy Self-Learning Controllers for Elasticity Management in Dynamic Cloud Architectures”. In: *2016 12th International ACM SIGSOFT Conference on Quality of Software Architectures (QoSA)*. Venice, Italy: IEEE, Apr. 2016, pp. 70–79. ISBN: 978-1-5090-2567-1. DOI: 10.1109/QoSA.2016.13. (Visited on 10/15/2024) (cit. on p. 27).

[KCH09] E. Kalyvianaki, T. Charalambous, S. Hand. “Self-Adaptive and Self-Configured CPU Resource Provisioning for Virtualized Servers Using Kalman Filters”. In: *Proceedings of the 6th International Conference on Autonomic Computing*. ICAC ’09. New York, NY, USA: Association for Computing Machinery, June 2009, pp. 117–126. ISBN: 978-1-60558-564-2. DOI: 10.1145/1555228.1555261. (Visited on 06/18/2024) (cit. on pp. 12–18, 60).

[KGM10] S. Khatua, A. Ghosh, N. Mukherjee. “Optimizing the Utilization of Virtual Resources in Cloud Environment”. In: *2010 IEEE International Conference on Virtual Environments, Human-Computer Interfaces and Measurement Systems*. Taranto, Italy: IEEE, Sept. 2010, pp. 82–87. ISBN: 978-1-4244-5904-9. DOI: 10.1109/VECIMS.2010.5609349. (Visited on 06/11/2024) (cit. on pp. 12–18, 59).

[KHK+17] J. V. Kistowski, N. Herbst, S. Kounev, H. Groenda, C. Stier, S. Lehrig. “Modeling and Extracting Load Intensity Profiles”. In: *ACM Trans. Auton. Adapt. Syst.* 11.4 (Jan. 2017), 23:1–23:28. ISSN: 1556-4665. DOI: 10.1145/3019596. (Visited on 11/09/2024) (cit. on pp. 35, 46).

[KKSB23] F. Klinaku, J. Katić, S. S. Stieß, S. Becker. “Designing Elasticity Policies for Cloud-Native Applications with Slingshot”. In: *2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. Västerås, Sweden: IEEE, Oct. 2023, pp. 19–23. ISBN: 9798350324983. DOI: 10.1109/MODELS-C59198.2023.00012. (Visited on 04/03/2024) (cit. on pp. iii, v, 1, 3, 7).

[LML14] T. Lorido-Botran, J. Miguel-Alonso, J. A. Lozano. “A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments”. In: *Journal of Grid Computing* 12.4 (Dec. 2014), pp. 559–592. ISSN: 1570-7873, 1572-9184. DOI: 10.1007/s10723-014-9314-7. (Visited on 04/20/2024) (cit. on pp. 1, 7, 9–11, 44).

[LPD+12] W. Lloyd, S. Pallickara, O. David, J. Lyon, M. Arabi, K. Rojas. “Performance Modeling to Support Multi-tier Application Deployment to Infrastructure-as-a-Service Clouds”. In: *2012 IEEE Fifth International Conference on Utility and Cloud Computing*. Chicago, IL, USA: IEEE, Nov. 2012, pp. 73–80. ISBN: 978-1-4673-4432-6 978-0-7695-4862-3. DOI: 10.1109/UCC.2012.20. (Visited on 06/11/2024) (cit. on pp. 12–18, 58).

[LZ10] P. Lama, X. Zhou. “Autonomic Provisioning with Self-Adaptive Neural Fuzzy Control for End-to-end Delay Guarantee”. In: *2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*. Miami Beach, FL, USA: IEEE, Aug. 2010, pp. 151–160. ISBN: 978-1-4244-8181-1. DOI: 10.1109/MASCOTS.2010.24. (Visited on 06/18/2024) (cit. on pp. 12–18, 59).

[MBE13] L. R. Moore, K. Bean, T. Ellahi. "Transforming Reactive Auto-Scaling into Proactive Auto-Scaling". In: *Proceedings of the 3rd International Workshop on Cloud Data and Platforms*. Prague Czech Republic: ACM, Apr. 2013, pp. 7–12. ISBN: 978-1-4503-2075-7. DOI: 10.1145/2460756.2460758. (Visited on 06/11/2024) (cit. on pp. 12–18, 20, 58).

[MBL+13] F. J. A. Morais, F. V. Brasileiro, R. V. Lopes, R. Araujo Santos, W. Satterfield, L. Rosa. "Autoflex: Service Agnostic Auto-scaling Framework for IaaS Deployment Models". In: *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*. May 2013, pp. 42–49. DOI: 10.1109/CCGrid.2013.74. (Visited on 06/17/2024) (cit. on pp. 12–18, 58).

[Men95] J. Mendel. "Fuzzy Logic Systems for Engineering: A Tutorial". In: *Proceedings of the IEEE* 83.3 (Mar. 1995), pp. 345–377. ISSN: 00189219. DOI: 10.1109/5.364485. (Visited on 11/21/2024) (cit. on p. 5).

[Mic24] Microsoft. *Use Predictive Autoscale to Scale out before Load Demands in Virtual Machine Scale Sets - Azure Monitor*. https://learn.microsoft.com/en-us/azure/azure-monitor/autoscale/autoscale-predictive. Nov. 2024. (Visited on 12/08/2024) (cit. on p. 1).

[MMZV13] R. Mian, P. Martin, F. Zulkernine, J. L. Vazquez-Poletti. "Towards Building Performance Models for Data-Intensive Workloads in Public Clouds". In: *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*. Prague Czech Republic: ACM, Apr. 2013, pp. 259–270. ISBN: 978-1-4503-1636-1. DOI: 10.1145/2479871.2479908. (Visited on 06/11/2024) (cit. on pp. 12–18, 58).

[MWY+10] H. Mi, H. Wang, G. Yin, Y. Zhou, D. Shi, L. Yuan. "Online Self-Reconfiguration with Performance Guarantee for Energy-Efficient Large-Scale Cloud Computing Data Centers". In: *2010 IEEE International Conference on Services Computing*. Miami, FL, USA: IEEE, July 2010, pp. 514–521. ISBN: 978-1-4244-8147-7. DOI: 10.1109/SCC.2010.69. (Visited on 06/11/2024) (cit. on pp. 12–18, 20, 59).

[NLV+19] S. M. R. Nouri, H. Li, S. Venugopal, W. Guo, M. He, W. Tian. "Autonomic Decentralized Elasticity Based on a Reinforcement Learning Controller for Cloud Applications". In: *Future Generation Computer Systems* 94 (May 2019), pp. 765–780. ISSN: 0167-739X. DOI: 10.1016/j.future.2018.11.049. (Visited on 06/12/2024) (cit. on pp. 12–19, 58).

[PHS+09] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant. "Automated Control of Multiple Virtualized Resources". In: *Proceedings of the 4th ACM European Conference on Computer Systems*. EuroSys '09. New York, NY, USA: Association for Computing Machinery, Apr. 2009, pp. 13–26. ISBN: 978-1-60558-482-9. DOI: 10.1145/1519065.1519068. (Visited on 06/18/2024) (cit. on pp. 12–18, 59).

[PN09] R. Prodan, V. Nae. "Prediction-Based Real-Time Resource Provisioning for Massively Multiplayer Online Games". In: *Future Generation Computer Systems* 25.7 (July 2009), pp. 785–793. ISSN: 0167-739X. DOI: 10.1016/j.future.2008.11.002. (Visited on 06/20/2024) (cit. on pp. 12–18, 59).

[RBX+09] J. Rao, X. Bu, C.-Z. Xu, L. Wang, G. Yin. "VCONF: A Reinforcement Learning Approach to Virtual Machines Auto-Configuration". In: *Proceedings of the 6th International Conference on Autonomic Computing*. ICAC '09. New York, NY, USA: Association for Computing Machinery, June 2009, pp. 137–146. ISBN: 978-1-60558-564-2. DOI: 10.1145/1555228.1555263. (Visited on 06/18/2024) (cit. on pp. 12–18, 60).

[RBXW11] J. Rao, X. Bu, C.-Z. Xu, K. Wang. "A Distributed Self-Learning Approach for Elastic Provisioning of Virtualized Cloud Resources". In: *2011 IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*. Singapore, Singapore: IEEE, July 2011, pp. 45–54. ISBN: 978-1-4577-0468-0. DOI: 10.1109/MASCOTS.2011.47. (Visited on 06/11/2024) (cit. on pp. 12–18, 59).

[RDG11] N. Roy, A. Dubey, A. Gokhale. "Efficient Autoscaling in the Cloud Using Predictive Models for Workload Forecasting". In: *2011 IEEE 4th International Conference on Cloud Computing*. Washington, DC, USA: IEEE, July 2011, pp. 500–507. ISBN: 978-1-4577-0836-7. DOI: 10.1109/CLOUD.2011.42. (Visited on 06/11/2024) (cit. on pp. 12–18, 59).

[SB20] R. S. Sutton, A. Barto. *Reinforcement Learning: An Introduction*. Second edition. Adaptive Computation and Machine Learning. Cambridge, Massachusetts London, England: The MIT Press, 2020. ISBN: 978-0-262-03924-6 (cit. on p. 4).

[SGJN19] P. Singh, P. Gupta, K. Jyoti, A. Nayyar. "Research on Auto-Scaling of Web Applications in Cloud: Survey, Trends and Future Directions". In: *Scalable Computing: Practice and Experience* 20.2 (May 2019), pp. 399–432. ISSN: 1895-1767. DOI: 10.12694/scpe.v20i2.1537. (Visited on 04/08/2024) (cit. on p. 7).

[SHE13] M. Sedaghat, F. Hernandez-Rodriguez, E. Elmroth. "A Virtual Machine Re-Packing Approach to the Horizontal vs. Vertical Elasticity Trade-off for Cloud Autoscaling". In: *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference*. CAC '13. New York, NY, USA: Association for Computing Machinery, Aug. 2013, pp. 1–10. ISBN: 978-1-4503-2172-3. DOI: 10.1145/2494621.2494628. (Visited on 06/17/2024) (cit. on pp. 12–18, 58).

[SSGW11] Z. Shen, S. Subbiah, X. Gu, J. Wilkes. "CloudScale: Elastic Resource Scaling for Multi-Tenant Cloud Systems". In: *Proceedings of the 2nd ACM Symposium on Cloud Computing*. SOCC '11. New York, NY, USA: Association for Computing Machinery, Oct. 2011, pp. 1–14. ISBN: 978-1-4503-0976-9. DOI: 10.1145/2038916.2038921. (Visited on 06/20/2024) (cit. on pp. 12–18, 59).

[SSSS11] U. Sharma, P. Shenoy, S. Sahu, A. Shaikh. "A Cost-Aware Elasticity Provisioning System for the Cloud". In: *2011 31st International Conference on Distributed Computing Systems*. June 2011, pp. 559–570. DOI: 10.1109/ICDCS.2011.59. (Visited on 06/13/2024) (cit. on pp. 12–18, 59).

[Ste09] D. Steinberg, ed. *EMF: Eclipse Modeling Framework*. 2nd ed., Rev. and updated. The Eclipse Series. Upper Saddle River, NJ: Addison-Wesley, 2009. ISBN: 978-0-321-33188-5 (cit. on p. 4).

[Szy99] C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. ACM Press Books. New York: ACM-Press [u.a.], 1999. ISBN: 978-0-201-17888-3 (cit. on p. 3).

[TJDB06] G. Tesauro, N. Jong, R. Das, M. Bennani. "A Hybrid Reinforcement Learning Approach to Autonomic Resource Allocation". In: *2006 IEEE International Conference on Autonomic Computing*. June 2006, pp. 65–73. DOI: 10.1109/ICAC.2006.1662383. (Visited on 06/13/2024) (cit. on pp. 12–18, 60).

[USC+08] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, T. Wood. "Agile Dynamic Provisioning of Multi-Tier Internet Applications". In: *ACM Transactions on Autonomous and Adaptive Systems* 3.1 (Mar. 2008), 1:1–1:39. ISSN: 1556-4665. DOI: 10.1145/1342171.1342172. (Visited on 06/18/2024) (cit. on pp. 12–18, 60).

[Van99] W. Van Der Aalst. "Formalization and Verification of Event-Driven Process Chains". In: *Information and Software Technology* 41.10 (July 1999), pp. 639–650. ISSN: 09505849. DOI: 10.1016/S0950-5849(99)00016-6. (Visited on 09/26/2024) (cit. on p. xv).

[VB21] S. Verma, A. Bala. "Auto-Scaling Techniques for IoT-based Cloud Applications: A Review". In: *Cluster Computing* 24.3 (Sept. 2021), pp. 2425–2459. ISSN: 1386-7857, 1573-7543. DOI: 10.1007/s10586-021-03265-9. (Visited on 04/08/2024) (cit. on p. 7).

[WGOI17] Z. Wang, C. Gwon, T. Oates, A. Iezzi. *Automated Cloud Provisioning on AWS Using Deep Reinforcement Learning*. Sept. 2017. arXiv: 1709.04305 [cs]. (Visited on 06/12/2024) (cit. on pp. 12–19, 58).

[WKL+19] Y. Wei, D. Kudenko, S. Liu, L. Pan, L. Wu, X. Meng. "A Reinforcement Learning Based Auto-Scaling Approach for SaaS Providers in Dynamic Cloud Environment". In: *Mathematical Problems in Engineering* 2019.1 (2019), p. 5080647. ISSN: 1563-5147. DOI: 10.1155/2019/5080647. (Visited on 06/12/2024) (cit. on pp. 12–19, 58).

[WXZ+11] L. Wang, J. Xu, M. Zhao, Y. Tu, J. A. Fortes. "Fuzzy Modeling Based Resource Management for Virtualized Database Systems". In: *2011 IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*. Singapore, Singapore: IEEE, July 2011, pp. 32–42. ISBN: 978-1-4577-0468-0. DOI: 10.1109/MASCOTS.2011.70. (Visited on 06/18/2024) (cit. on pp. 12–18, 59).

[XJC13] Xiangping Bu, Jia Rao, Cheng-Zhong Xu. "Coordinated Self-Configuration of Virtual Machines and Appliances Using a Model-Free Learning Approach". In: *IEEE Transactions on Parallel and Distributed Systems* 24.4 (Apr. 2013), pp. 681–690. ISSN: 1045-9219. DOI: 10.1109/TPDS.2012.174. (Visited on 06/18/2024) (cit. on pp. 12–18, 58).

[XRB12] C.-Z. Xu, J. Rao, X. Bu. "URL: A Unified Reinforcement Learning Approach for Autonomic Cloud Management". In: *Journal of Parallel and Distributed Computing* 72.2 (Feb. 2012), pp. 95–105. ISSN: 0743-7315. DOI: 10.1016/j.jpdc.2011.10.003. (Visited on 06/11/2024) (cit. on pp. 12–18, 59).

[YLS+14] J. Yang, C. Liu, Y. Shang, B. Cheng, Z. Mao, C. Liu, L. Niu, J. Chen. "A Cost-Aware Auto-Scaling Approach Using the Workload Prediction in Service Clouds". In: *Information Systems Frontiers* 16.1 (Mar. 2014), pp. 7–18. ISSN: 1572-9419. DOI: 10.1007/s10796-013-9459-0. (Visited on 06/17/2024) (cit. on pp. 12–18, 58).

[ZA12] Q. Zhu, G. Agrawal. "Resource Provisioning with Budget Constraints for Adaptive Applications in Cloud Environments". In: *IEEE Transactions on Services Computing* 5.4 (2012), pp. 497–511. ISSN: 1939-1374. DOI: 10.1109/TSC.2011.61. (Visited on 06/18/2024) (cit. on pp. 12–18, 59).

[ZCS07] Q. Zhang, L. Cherkasova, E. Smirni. "A Regression-Based Analytic Model for Dynamic Resource Provisioning of Multi-Tier Applications". In: *Fourth International Conference on Autonomic Computing (ICAC'07)*. Jacksonville, FL, USA: IEEE, June 2007, pp. 27–27. ISBN: 978-0-7695-2779-6. DOI: 10.1109/ICAC.2007.1. (Visited on 06/18/2024) (cit. on pp. 12–18, 60).

[ZXW10] Zhenhuan Gong, Xiaohui Gu, J. Wilkes. "PRESS: PRedictive Elastic ReSource Scaling for Cloud Systems". In: *2010 International Conference on Network and Service Management*. Niagara Falls, ON, Canada: IEEE, Oct. 2010, pp. 9–16. ISBN: 978-1-4244-8910-7. DOI: 10.1109/CNSM.2010.5691343. (Visited on 06/11/2024) (cit. on pp. 12–18, 59).

All links were last followed on July 2, 2024.

A Detailed results of the literature review

A comparison of all approaches can be found in Table A.1.

The used criteria are discussed in Section 4.2. A short overview of the approaches that fulfill the criteria for inclusion inside the Slingshot approach is provided in Section 4.3.

Approach	Prediction Type		Approach Performance	Simulation Benefit	Ease of Implementation	Evaluation	Modeling Difficulty	Scaling Dimension
	Workload	Resource						
[WKL+19]	-	✔	slow training / convergence	/	possible	✔	cost	horizontal
[NLV+19]	-	✔	good	/	possible	✔	cost	horizontal
[BD19]	-	✔	good	/	possible	✔	okay	horizontal
[GJP18]	✔	✔	good	/	possible	-	okay	horizontal
[HA18]	✔	✔	good	model transfer	possible	✔	okay	horizontal
[DS18]	-	✔	good	/	possible	✔	architecture	horizontal
[APJE17]	-	✔	good	model transfer	possible	✔	okay	horizontal
[WGOI17]	-	✔	slow training / convergence	model transfer	possible	✔	cost	horizontal
[YLS+14]	✔	✗	good	parameter tuning	possible	-	okay	horizontal and vertical
[SHE13]	-	✔	missing detail	/	difficult	-	cost	horizontal
[CCG13]	-	✔	okay	parameter tuning	difficult	✗	okay	horizontal
[BHD13]	-	✔	okay	/	possible	-	cost	horizontal
[MBL+13]	✔	✔	good	simulation	possible	✗	okay	horizontal
[CS13]	✗	✗	N/A	/	possible	✔	architecture	horizontal
[JPLC13]	✔	-	okay	/	difficult	✗	cost	horizontal
[XJC13]	-	✔	good	/	possible	✔	okay	vertical
[MMZV13]	✗	✔	N/A	/	no scaling	-	no scaling	none
[MBE13]	✔	✔	good	/	possible	✔	okay	horizontal
[ATE12]	✔	✔	good	parameter tuning	possible	✔	okay	horizontal
[AKTE12]	✔	✔	okay	parameter tuning	possible	✔	architecture	horizontal
[IKLL12]	✔	✗	okay	/	no scaling	✗	no scaling	none
[GT12]	✗	✔	missing detail	/	missing details	✗	okay	horizontal
[LPD+12]	✗	✔	N/A	/	no scaling	-	no scaling	none

[FLWC12]	✔	✔	N/A	/	difficult	✗	okay	horizontal and vertical
[ZA12]	✔	✔	good	/	difficult	-	okay	vertical
[HLY12]	✔	-	N/A	/	no scaling	-	no scaling	none
[DGVV12]	✔	✔	good	parameter tuning	difficult	✗	cost	horizontal and vertical
[XRB12]	-	✔	okay	model transfer	offline training	-	metric detail	vertical
[SSSS11]	✗	✗	N/A	/	difficult	✗	no scaling	horizontal
[RBXW11]	-	✔	good	model transfer	offline training	-	metric detail	vertical
[IDCJ11]	-	✔	worse than other approaches	/	possible	-	okay	horizontal
[SSGW11]	✔	-	good	/	possible	-	okay	vertical
[RDG11]	✔	✔	okay	parameter tuning	possible	-	okay	horizontal
[WXZ+11]	✔	✔	okay	model transfer	possible	✔	okay	vertical
[CDM11]	✔	✗	okay	parameter tuning	no scaling	✗	okay	vertical
[CH11]	✗	✔	N/A	/	no scaling	✗	no scaling	none
[DKM+11]	-	✔	slow training / convergence	/	possible	-	cost	horizontal
[LZ10]	✔	✔	good	model transfer	difficult	-	okay	horizontal
[CDM10]	✔	✗	okay	parameter tuning	no scaling	✔	architecture	none
[DMM+10]	-	✔	good	model transfer	possible	✔	okay	horizontal
[MWY+10]	✔	-	good	/	difficult	✔	too slow	horizontal
[KGM10]	✔	✗	N/A	/	missing details	-	no scaling	horizontal and vertical
[ZXW10]	✔	-	good	parameter tuning	possible	✔	okay	vertical
[BVU+10]	✗	✔	N/A	/	difficult	✗	architecture	none
[PHS+09]	✔	✔	okay	parameter tuning	possible	-	cost	vertical
[PN09]	✔	✔	N/A	/	difficult	✗	architecture	horizontal

[KCH09]	-	✔	okay	/	difficult	✔	okay	vertical
[BGS+09]	✔	✔	okay	simulation	offline training	✗	okay	horizontal
[RBX+09]	-	✔	okay	/	possible	✔	metric detail	vertical
[USC+08]	✔	✗	okay	/	offline training	-	architecture	horizontal
[CHL+08]	✔	✔	worse than other approaches	/	offline training	✗	okay	horizontal
[ZCS07]	✗	✔	good	/	no scaling	✗	no scaling	none
[TJDB06]	-	✔	okay	/	difficult	-	architecture	none
[CGS03]	✔	✔	okay	parameter tuning	difficult	✔	metric detail	none

Table A.1: Classification for each approach

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature