# QoS-Based Optimization of Runtime Management of Sensing Cloud Applications

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der KIT-Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Manuel Gotin

aus Sindelfingen

Tag der mündlichen Prüfung: 30.04.2021
Erster Gutachter: Prof. Dr. Ralf Reussner
Zweiter Gutachter: Prof. Dr. Andreas Oberweis

# Abstract

This thesis contributes to designing and operating IoT applications by introducing Quality-of-Service (QoS) cost functions and time-driven flow control approaches.

IoT applications perceive the environment via sensors of smart devices in order to analyze the environment or interact with it. Smart devices are limited in terms of processing and storage capabilities. Therefore, many IoT applications connect the smart devices to elastic and scalable cloud services via IoT platforms. The load on the cloud service is induced by the connected smart devices, which continuously transfer messages containing data about the environmental state. The resource configuration of the cloud service influences its capacity. Therefore, a service operator, who operates an IoT application, is faced with the challenge of configuring the smart devices and the cloud service in a manner, which achieves a high data quality at low operational costs.

To support the service operator at design time, we model cost functions for data qualities, which are influenced by the interplay of smart device and cloud service configurations. These cost functions enable a service operator to search for a cost minimal configuration for specific scenarios. Existing approaches for optimizing applications at design time focus on traditional software architectures and therefore do not provide necessary concepts to cost model qualities of IoT applications.

We also support the service operator with flow control approaches that cope with capacity shortages of cloud services by reducing the message rate of smart devices in a controlled manner. While this can have a negative impact on the accuracy of measurements, it stabilizes processing delays and ensures, that the IoT application remains available in severe overload scenarios. Existing runtime approaches focus on an automatic resource provisioning of cloud services by auto-scalers. Whereas they enable to cope with capacity shortages and load variations by levering the elasticity of the cloud environment, the achieved QoS can be associated with high operating costs. In contrast, the presented flow control approaches reduce the load on the cloud service and probe their capacity. Therefore, they enable to efficiently use the capacity of a cloud service and to cope with capacity shortages. Existing approaches adapting smart devices focus on qualities such as accuracy or energy efficiency and are therefore unsuitable to cope with overload scenarios.

In summary, the thesis contributes the following:

- **Evaluation of performance metrics for scaling decisions.** We have evaluated infrastructure- and application-level metrics in regard to their suitability for scaling microservices with varying resource characteristics. Based on the results, a service operator can make an educated decision to select a suitable performance metric for an auto-scaler of a specific microservice.

- **Design of QoS cost functions for IoT applications.** We present a cost model that captures the impact of smart device and resource configurations on the qualities of IoT applications. Based on these cost models, the configuration of IoT applications can be optimized at design time. Furthermore, the cost functions can be used to evaluate QoS contributions of runtime management approaches.

- **Design of flow control approaches for IoT applications.** The presented approaches offer a complementary mechanism to auto-scaling to maintain the QoS in overload scenarios. The overall load on the cloud service is reduced by adjusting the message rate of the smart devices. This enables a service operator to counter a cloud services' capacity shortage by degrading data qualities.
- **Coupling of flow control approaches with auto-scaling.** We present rule-based coupling mechanisms that activate flow control or auto-scalers in a reactive manner. The coupling aims improve the overall QoS conformance by managing capacity shortages and load variations with a combination of resource provisioning and data quality reductions.
- **Design of a framework for developing self-adaptive systems.** The self-adaptive framework provides an application model for IoT applications and concepts for the reconfiguration of microservices and smart devices. It supports heterogeneous cloud environments and accelerates the prototypical development of runtime management approaches.

In order to empirically answer the research questions, we have instantiated the presented framework for self-adaptive IoT systems to implement and deploy flow control and auto-scaling approaches. For each case study system and application scenario, we have instantiated the cost functions and integrated them into an optimization framework to search for cost minimal runtime management configurations. We validated the approaches using two case study systems of varying complexity. The first case study system consists of a cloud service that processes messages from virtual smart devices via an IoT platform. With this system we analyzed the characteristics of the presented flow control approaches in different application scenarios and compared them to auto-scaling and a coupling of the approaches. The results showed, that the flow control approaches can address overload scenarios as efficiently as auto-scaling and that the emerging QoS is in a comparable range. On average, the flow control approaches achieved about 50 % lower total QoS costs in the investigated scenarios. Both auto-scaling and flow control had significant disadvantages in certain application scenarios, e.g. when data accuracy or resource costs are the primary concern. In all cases, a coupling had resulted in lower QoS costs. In the second case study we implemented an intelligent heating solution from the Robert Bosch GmbH to validate the approaches on a multi-service IoT application. Again, a combination of flow control and auto-scaling has resulted in a high data quality at low resource costs. Overall, the results showed that a runtime management with the presented flow control approaches is beneficial for the QoS of IoT applications.

# Zusammenfassung

Die vorliegende Arbeit präsentiert Ansätze und Techniken zur qualitätsbewussten Verbesserung des Laufzeitmanagements von IoT-Anwendungen.

IoT-Anwendungen nehmen über die Sensorik von Smart Devices ihre Umgebung wahr, um diese zu analysieren oder mit ihr zu interagieren. Smart Devices sind in der Rechen- und Speicherleistung begrenzt, weshalb viele IoT-Anwendungen über eine IoT Plattform mit elastischen und skalierbaren Cloud Services verbunden sind. Die Last auf dem Cloud Service entsteht durch die verbundenen Smart Devices, die kontinuierlich Nachrichten transferieren. Die Ressourcenkonfiguration des Cloud Services beeinflusst dessen Kapazität. Ein Service Operator, der eine IoT-Anwendung betreibt, ist mit der Herausforderung konfrontiert, die Smart Devices und den Cloud Service so zu konfigurieren, dass eine hohe Datenqualität bei niedrigen Betriebskosten erreicht wird.

Um hierbei den Service Operator zur Design Time zu unterstützen, modellieren wir Kostenfunktionen für Datenqualitäten, die durch das Wechselspiel der Smart Device- und Cloud Service-Konfiguration beeinflusst werden. Mit Hilfe dieser Kostenfunktionen kann ein Service Operator nach einer kostenminimalen Konfiguration für bestimmte Szenarien suchen. Existierende Ansätze zur Optimierung von Anwendungen zur Design Time fokussieren sich auf traditionelle Software-Architekturen und bieten daher nicht die notwendigen Konzepte zur Kostenmodellierung von IoT-Anwendungen an.

Des Weiteren unterstützen wir den Service Operator durch Lastkontrollverfahren, die auf Kapazitätsengpässe des Cloud Services durch eine kontrollierte Reduktion der Nachrichtenrate reagieren. Während sich das auf die Genauigkeit der Messungen nachteilig auswirken kann, stabilisieren sich zeitliche Verzögerungen und die IoT-Anwendung bleibt auch in starken Überlastszenarien verfügbar. Existierende Laufzeittechniken fokussieren sich auf die automatische Ressourcenprovisionierung von Cloud Services durch Auto-Scaler. Diese ermöglichen zwar, auf Kapazitätsengpässe und Lastschwankungen zu reagieren, doch die erreichte Quality-of-Service (QoS) kann dadurch mit hohen Betriebskosten verbunden sein. Daher ermöglichen wir durch die Lastkontrollverfahren eine weitere Technik, mit der einerseits dynamisch auf Kapazitätsengpässe reagiert werden und andererseits die zur Verfügung stehende Kapazität eines Cloud Services effizient genutzt werden kann. Außerdem präsentieren wir Kopplungstechniken, die Auto-Scaling und Lastkontrollverfahren kombinieren. Bestehende Ansätze zur Rekonfiguration von Smart Devices konzentrieren sich auf Qualitäten wie Genauigkeit oder Energie-Effizienz und sind daher ungeeignet, um auf Kapazitätsengpässe zu reagieren.

Zusammenfassend liefert die Dissertation die folgenden Beiträge:

- **Untersuchung von Performance Metriken für Skalierentscheidungen.** Wir haben Infrastuktur- und Anwendungsebenen-Metriken daraufhin evaluiert, wie geeignet sie für Skalierentscheidungen von Microservices sind, die variierende

Charakteristiken aufweisen. Auf Basis der Ergebnisse kann ein Service Operator eine fundierte Entscheidung darüber treffen, welche Performance Metrik zur Skalierung eines bestimmten Microservices am geeignesten ist.

- **Design von QoS Kostenfunktionen für IoT-Anwendungen.** Wir haben ein QoS Kostenmodell aufgestellt, dass das Wirken von Smart Device- und Cloud Service-Konfiguration auf die Qualitäten einer IoT-Anwendung erfasst. Auf Grundlage dieser Kostenmodelle kann die Konfiguration von IoT-Anwendungen zur Design Time optimiert werden. Des Weiteren können mit den Kostenfunktionen Laufzeitverfahren hinsichtlich ihrem Beitrag zur QoS für verschiedene Szenarien evaluiert werden.
- **Entwicklung von Lastkontrollverfahren für IoT-Anwendungen.** Die präsentierten Verfahren bieten einen komplementären Mechanismus zu Auto-Scaling an, um bei Kapazitätsengpässen die QoS aufrechtzuerhalten. Hierbei wird die Gesamtlast auf dem Cloud Service durch Anpassungen der Nachrichtenrate der Smart Devices reduziert. Ein Service Operator hat hiermit die Möglichkeit, Kapazitätsengpässen über eine Degradierung der Datenqualität zu begegnen.
- **Kopplung von Lastkontrollverfahren mit Ressourcen-Provisionierung.** Wir präsentieren regelbasierte Kopplungsmechanismen, die reaktiv Lastkontrollverfahren oder Auto-Scaler aktivieren und diese damit koppeln. Das ermöglicht, auf Kapazitätsengpässe über eine Kombination von Datenqualitätsreduzierungen und Ressourcekostenerhöhungen zu reagieren.
- **Design eines Frameworks zur Entwicklung selbst-adaptiver Systeme.** Das selbst-adaptive Framework bietet ein Anwendungsmodell für IoT-Anwendungen und Konzepte für die Rekonfiguration von Microservices und Smart Devices an. Es kann in verschiedenen Cloud-Umgebungen aufgesetzt werden und beschleunigt die prototypische Entwicklung von Laufzeitverfahren.

Wir validierten die Ansätze anhand zweier Case Study Systeme unterschiedlicher Komplexität. Das erste Case Study System besteht aus einem Cloud Service, welcher über eine IoT Plattform Nachrichten von virtuellen Smart Devices verarbeitet. Mit diesem System haben wir für unterschiedliche Anwendungsszenarien die Charakteristiken der vorgestellten Lastkontrollverfahren analysiert, um diese gegen Auto-Scaling und einer Kopplung der Ansätze zu vergleichen. Hierbei stellte sich heraus, dass die Lastkontrollverfahren ähnlich effizient wie Auto-Scaler Überlastszenarien addressieren können und sich die QoS in einem vergleichbaren Bereich bewegt. Im Schnitt erreichten die Lastkontrollverfahren in den untersuchten Szenarien etwa 50 % geringere QoS Gesamtkosten. Es zeigte sich auch, dass sowohl Auto-Scaling als auch die Lastkontrollverfahren in bestimmten Anwendungsszenarien deutliche Nachteile haben, so z. B. wenn die Datengenauigkeit oder Ressourcenkosten im Vordergrund stehen. Es hat sich gezeigt, dass eine Kopplung hierbei immer vorteilhaft ist, um die QoS beizubehalten. Im zweiten Case Study System haben wir eine intelligente Heizungslösung der Robert Bosch GmbH implementiert, um die Ansätze an einem komplexeren System zu validieren. Auch hier zeigte sich, dass eine Kombination von Lastkontrolle und Auto-Scaling am vorteilhaftesten ist und zu einer hohen Datenqualität bei geringen Ressourcenkosten beiträgt.

Die Ergebnisse zeigen, dass die vorgestellten Lastkontrollverfahren geeignet sind, die QoS von IoT Anwendungen zu verbessern. Es bietet einem Service Operator damit ein weiteres Werkzeug für das Laufzeitmanagement von IoT Anwendungen, dass einen zum Auto-Scaling komplementären Mechanismus verwendet. Das hier vorgestellte Framework zur Entwicklung selbst-adaptiver IoT Systeme haben wir zur empirischen Beantwortung der Forschungsfragen instanziiert und damit dessen Eignung demonstriert. Wir zeigen außerdem eine exemplarische Verwendung der vorgestellten Kostenfunktionen für verschiedene Anwendungsszenarien und binden diese im Zuge der Validierung in einem Optimierungs-Framework ein.

# Danksagungen

Diese Dissertation wäre ohne die Unterstützung vieler Menschen nicht möglich gewesen. Während meiner Doktorandenzeit beim Forschungscampus der Robert Bosch GmbH und am Karlsruher Institut für Technologie (KIT) hatte ich die Möglichkeit, Forschung sowohl in der Industrie als auch an der Universität kennenzulernen. Hierbei bin ich vielen Menschen begegnet, die mich auf unterschiedlichste Weise während dieser herausfordernden Zeit unterstützt haben, und denen ich an dieser Stelle danken möchte.

Zunächst einmal gilt mein herzlicher Dank meinem Doktorvater Prof. Dr. Ralf H. Reussner, der mich als Doktorand in seiner Forschungsgruppe aufgenommen hat. Hier konnte ich in den Genuss einer hervorragenden Betreuung kommen und Teil einer harmonischen und äußerst fruchtbaren Arbeitsatmosphäre werden. Ich danke auch herzlich Herrn Prof. Dr. Andreas Oberweis für die Übernahme des Korreferats. Ebenso danke ich Frau Prof. Dr. Anne Koziolek für die fachlichen Ratschläge und Herrn Dr. Robert Heinrich für die Unterstützung bei Publikationen. Meinem Kollegen Dominik Werle vom KIT möchte ich für die gute Zusammenarbeit danken, die stets sehr produktiv und zielführend war.

Mein herzliches Dank gilt ebenso meinem Betreuer bei der Robert Bosch GmbH, Herrn Dr. Felix Lösch, der mir stets durch fachliche und methodische Ratschläge eine große Unterstützung war. Ebenso danke ich meinem Gruppenleiter Herr Dr. Dirk Ziegenbein für die personelle Betreuung. Ich danke meiner Arbeitskollegin Frau Dr. Julia Leibinger für die fachlichen Diskussionen und die Unterstützung bei Promotionsfragen. Den Mitdoktoranden bei der Robert Bosch GmbH danke ich für das Gemeinschaftsgefühl und die Aktivitäten außerhalb der Forschung.

Als sehr positiv empfand ich die Zusammenarbeit mit der Uni Stuttgart und dem Forschungszetrum für Informatik (FZI) im Rahmen eines öffentlich geförderten Projekts. Daher bedanke ich mich an dieser Stelle bei Herrn Prof. Dr. Steffen Becker und Floriment Klinaku aus der Uni Stuttgart sowie Jörg Henß und Martina Rapp vom FZI. Mein Dank gilt ebenso allen Doktoranden von Herrn Prof. Reussner und Frau Prof. Koziolek, die mich als externen Doktoranden in ihrer Gruppe stets willkommen geheißen haben. Die gemeinsamen Klausurtagungen werde ich als lehrreich und schön in Erinnerung behalten. Ich bedanke mich außerdem bei den von mir betreuten Studenten Philipp Lehr, Niko Benkler und Daniel Handloser für die gute Zusammenarbeit.

Mein besonderer Dank gilt meinen Eltern, die meine Interessen von früher Kindheit an gefördert haben. Ebenso möchte ich meiner Schwester Nathalie danken, die mir seit jeher Rückhalt in allen Lebensbereichen bietet. Eure Unterstützung über all die Jahre hat mir das alles ermöglicht.

# Contents

# List of Figures

# List of Tables

# 1. Introduction

With the uprising of Internet-of-Things (IoT) the Robert Bosch GmbH is in the process of a large transformation in its application and infrastructure landscape. By offering many products ranging from smart home, industry 4.0 to smart devices Bosch is faced with the inherent challenges of operating *sensing cloud applications*, which provide ubiquitous access to sensor data in order to analyze the environment or actuate with it.

This thesis contributes to designing and operating sensing cloud applications by introducing Quality-of-Service (QoS) cost functions and time-driven flow control approaches. The cost functions enable to make the resource configuration of sensing cloud application QoS cost optimal by applying optimization methods. The flow control approaches aim to maintain the QoS in overload situations on the expense of data accuracy by adjusting the message transmission rate of smart devices.

This chapter illustrates why time-driven flow control approaches are important in managing sensing cloud applications. We identify a gap in state of the art that concerns flow control approaches for sensing cloud applications. We derive a set of challenges and research questions. Furthermore, we elaborate the contributions of the thesis and conclude it with an outline.

## 1.1. Motivation

The uprising of Cloud Computing and the IoT has resulted in a big shift in the software landscape. Cloud computing offers a flexible infrastructure to provision resources on demand with the illusion of unlimited computation and storage capabilities. The IoT aims to interconnect everyday devices in order to sense the physical environment and to interact with it. Since devices are limited in terms of computation and storage capabilities, cloud computing complements the IoT.

Due to the complementary characteristics of IoT and cloud computing their integration benefits many application scenarios and enables new smart services [13]. This thesis focuses on *Sensing-as-a-Service* and *Sensing-and-Actuation-as-a-Service*, which are part of many use cases, e.g. smart home or connected vehicles. Whereas the first aims to provide ubiquitous access to sensor data, the second aims to enable automatic control logic in the cloud. We refer to these applications as sensing cloud applications.

Sensing cloud applications can experience many dynamics. Resources of a cloud infrastructure are prone to capacity variations in computation [24] or network [60][59] resulting in changes of the performance characteristics. The number of connected devices can vary greatly, due to periodic and non-periodic fluctuations, e.g, weekend usage. Furthermore, environmental changes may result in changes of the sensing interval. It is challenging to capture the dynamics in design time, which makes it difficult to plan the resource configuration ahead.

Runtime management systems enable to cope with these dynamics by reacting to changes with reconfigurations. A widespread method in cloud computing is auto-scaling, which reacts to load by provisioning resources in an autonomous manner. They affect the elasticity of cloud applications, which refers to the degree to which a system is able to adapt to workload changes [37]. Therefore, they induce resource costs in order to maintain the QoS. However, in some scenarios resource provisioning is not a feasible option, based on operational or economical constraints. If the load produced by the smart devices exceeds the processing rate of the cloud solution, messages accumulate in the messaging infrastructure. Such an imbalance eventually results in an overload situation which threatens the QoS by degrading the latency, jitter and availability of the services. In the current state of the practice, a service provider has to carefully prepare the message broker in order to cope with such situations, e.g. by discarding messages or halting message producers if a threshold is exceeded, e.g. memory contention. However, both techniques have a high impact on the QoS by potentially inducing a high latency, jitter or message loss. Therefore, we identified a lack of approaches, which control the load on sensing cloud applications by reconfiguring smart devices.

Runtime management approaches impact the QoS in multiple dimensions. Resource provisioning might increases the costs of operating a system but avoids unacceptable response times. Adjusting the sensing rate of smart devices might decreases the sensing quality but avoids resource costs. Service operators have to decide which configuration meets contradictory QoS goals to an acceptable extent. Therefore it is important to understand which qualities of sensing cloud applications are affected by the runtime management approaches. Identifying the qualities enables a service operator to search for optimal runtime management configurations for sensing cloud applications.

The goal of this thesis is to enable an optimization of runtime management of sensing cloud applications based on QoS cost functions. Additionally, it aims to enable the performance runtime management of sensing cloud applications by smart device reconfigurations.

## 1.2. Problem Statement

We identified the following problem areas.

**Insufficient Consideration of Microservice Characteristics for Auto-Scaling.** The elasticity of cloud services is leveraged by auto-scaling. Threshold-based rules auto-scalers make scaling decisions based on a performance metric and preconfigured thresholds. Many implementations do not consider the resource requirements of the microservice and rely on low-level system metrics, like the CPU. There is a lack of understanding which performance metric and microservice combination achieves the highest elasticity. Existing approaches to evaluate the elasticity of threshold-based auto-scalers lack a model which is able to express resource characteristics of microservices [56] or do not provide message queues as infrastructure components [10].

**Lack of Flow Control Approaches for Smart Devices.** The elasticity of a cloud service is challenged by a varying load induced by smart devices. If the cloud service is underprovisioned overload situations arise, which affects the performance and availability. Existing flow control approaches for smart devices focus on accuracy [67] or overload protection on infrastructure layer [54]. There is a lack of approaches for adjusting the behavior of smart devices to address overload situations.

**Insufficient Consideration of QoS Effects of Runtime Management.** Resource provisioning and transmission rate adaptations impact both the QoS of a sensing cloud application. It is important to understand the dimensions of a sensing cloud application and how they can be quantified. Such cost functions allow to reduce the configuration effort for runtime management systems, as they can be optimized by a search procedure.

**High Effort for Deploying Runtime Management Approaches.** Existing approaches do not target cloud environments [31] or miss representations for smart devices [3]. The lack of a design model for runtime management approaches for sensing cloud applications results in a high effort to prototypical develop them and to deploy them in heterogeneous environments.

## 1.3. State of the Art

The analysis of qualities of IoT applications has recently become an area of interest to many researchers [70].

There is currently a plethora of transport level protocols for wired and wireless networks. Approaches like [51] organize the network as multi-hop communication, in which each device can relay packages, enabling a reliable message delivery by alternative paths. Approaches like [27] enforce QoS by package forwarding and queue priority policies, thus allowing to prioritize data based on the applications QoS demand. Despite the standardization of application layer protocols to interact with IoT devices, e.g. MQ Telemetry Transport (MQTT), an enforcement of QoS requires explicit support from the application protocols. In the current state of the art, they focus on a reliable delivery and orderliness over the network without concepts for coping with cloud services experiences capacity shortages.

Current approaches reconfiguring the behavior of smart devices are focused on non-performance related qualities, e.g. battery-efficiency [52] or data quality [67]. Congestion control approaches are either tailored to network communication [36] or include infrastructure components, like load balancers [36], which is not feasible to be managed by a service operator. Existing auto-scaling approaches are able to maintain QoS guarantees by sophisticated mechanisms as presented in [6]. However, they can be faced with provisioning constraints based on economical or resource limitations.

Frameworks to develop and deploy runtime management approaches are mainly based on application models which do not consider the microservice architectural style [31] or smart devices [3]. Whereas this is mitigated by cloud platform providers by providing a framework to define custom logic and metrics for scaling decisions, it essentially locks a service operator to the technology stack and does not allow to reuse approaches on different cloud environments with minimal effort.

The analysis and optimization of applications has a long tradition in computer science. Whereas approaches like the Palladio component model [11] and SimuLizar [10] are able to analyze the qualities of a software system, both are insufficient to analyze the qualities of IoT applications. Whereas Palladio lacks support for self-adaptation rules, e.g. to model auto-scalers, SimuLizar and Palladio lacks concepts to model infrastructure components like message queues [47].

## 1.4. Challenges and Research Questions

A set of challenges have to be addressed to optimize the runtime management of sensing cloud applications. This section introduces the Research Questions (RQ) which we have addressed in this thesis.

### 1.4.1. Understanding Performance Metrics for Auto-Scaling

Threshold-based rules auto-scaling system are a popular auto-scaling technique provided by many cloud platform providers. It defines a set of rules, which consist of a threshold and the value of a performance metric. If the threshold is exceeded, a scale in or scale out operation is executed. Microservice can experience capacity variations based on the underlying cloud infrastructure or external systems, e.g. storage or cloud services. Microservices communicate with each other using lightweight communication protocols like REST or utilizing queues provided by a message broker. Whereas the performance metric is often based on infrastructure metrics like the CPU utilization modern message broker systems provide monitoring capabilities for message queue, allowing to leverage them as performance metrics for scaling decisions. Therefore it is challenging for a service operator to select performance metrics tailored to the characteristics of the microservice. The thesis addresses the following questions to model and understand scaling decisions based on the choice of the performance metric:

**Research Question 1.** *How robust are infrastructure metrics for scaling microservice to capacity and resource demand variations?*

**Research Question 1.1.** *How are the infrastructure metrics affected by capacity or resource demand variations?*

**Research Question 1.2.** *How do capacity or resource demand variations affect the elasticity of threshold-based auto-scaling systems?*

### 1.4.2. Modeling and Optimization of the QoS of Sensing Cloud Applications

The Cloud-IoT paradigms enables sensing cloud applications by combining the IoT with cloud computing. From the viewpoint of a service operator it is important to maintain data qualities in order to satisfy the customer but also to maintain the resource costs. Whereas the data qualities are determined by time and accuracy aspects of smart device streams, the resource costs are induced by leasing resources. In order to maintain the QoS runtime management approaches can reconfigure the system, e.g. by (de-)provisioning resources. IoT platforms which connect smart devices with cloud services provide the capability to communicate with smart devices in order to reconfigure their behavior, e.g. by transmission rate adjustments. In the current state there is no model of the QoS for sensing cloud applications in terms of the impact of the resource configuration and the transmission rate of a sensing cloud application. Therefore, this thesis derives the following RQ:

**Research Question 2.** *How to model the QoS conformance in terms of the impact of the resource configuration and the transmission rate of a sensing cloud application to enable an efficient resource management?*

Being able to quantify the QoS of a sensing cloud application enables to search for optimal resource and transmission rate configurations. Furthermore, it allows to evaluate runtime management approaches based on a specific scenario. Existing optimization approaches for software systems lack the concepts of sensing cloud applications. Therefore, this thesis discusses a design-time optimization framework:

**Research Question 2.1.** *How to model the QoS costs of a sensing cloud application?*

**Research Question 2.2.** *What is a sufficiently fast approach for optimizing the QoS conformance of a sensing cloud application?*

### 1.4.3. Flow Control Approaches for Smart Devices

Auto-scalers are a popular mechanism to runtime manage sensing cloud applications. In some scenarios resource provisioning is not a feasible option, based on operational or economical constraints. If the load produced by the smart devices exceeds the processing rate of the cloud solution, messages accumulate in the messaging infrastructure, degrading the QoS. The state of the art is concerned with the collection rate based on accuracy considerations. Therefore, flow control approaches, which adjust the load based on the available capacity can be feasible additional mechanism to maintain the QoS in overload scenarios. Congestion avoidance mechanisms are a well-known concept of transport-layer protocols. We deem it as a challenge to transfer these concepts to application-layer. We introduce the following questions:

**Research Question 3.** *What are suitable flow control approaches to improve the QoS conformance of sensing cloud applications?*

Runtime managing a sensing cloud application using only auto-scalers or flow control approaches, may degrade the QoS in terms of resource costs or data qualities too much. Balancing the management of capacity shortages on both mechanisms may result in a better QoS. Therefore, the thesis investigates the following research question:

**Research Question 4.** *What are suitable coupling mechanisms to improve the QoS conformance of sensing cloud applications by combining flow control with auto-scaling?*

### 1.4.4. Runtime Management of Sensing Cloud Applications

It can be challenging for a service operator to set up runtime management approaches. Many cloud platform providers offer auto-scalers but do not provide a general interface for strategies beyond auto-scaling. Depending on the cloud provider, there is also a differently powerful interface for configuring the auto-scaler. The vendor lock-in makes it difficult to set up a uniform and reusable set of runtime strategies, especially for hybrid cloud scenarios. Existing runtime management frameworks like Rainbow [31] mainly focus on traditional software systems and therefore often do not have concepts such as smart devices, message queues and microservices. Therefore, it is a challenge for the service operator to set up runtime management approaches, which is addressed with the following RQ:

**Research Question 5.** *What is a good abstraction level for modeling sensing cloud applications for a self-adaptive platform? We consider a model abstraction good if it can be operated on multiple cloud platforms and is conformant to existing architectural viewpoints.*

## 1.5. Contributions

The contributions of this thesis are:

**C1: Evaluation of the impact of capacity and resource demand variations on scaling decisions.** We obtained insight in selecting performance metrics by evaluating the influence of microservice characteristic variations on the elasticity of threshold-based rules auto-scaler. The elasticity of CPU-based auto-scalers is greatly decreased if the CPU share of processing messages decreases, e.g. due to an increased wait time or computation capacity. Queue-based auto-scalers outperforms CPU-based auto-scalers for scaling microservices with varying wait time. Additionally, the presented model allows to predict the achieved elasticity of an auto-scaler based on the CPU or message queue metrics.

**C2: Design of QoS cost functions for sensing cloud applications and a QoS optimization framework.** The proposed QoS cost functions are able to capture the effect of resource and transmission rate reconfigurations on the qualities of sensing cloud applications. The presented optimization framework allows to integrate a simulation model and optimization method to search for runtime or resource configuration candidates to minimize the cumulative QoS costs.

**C3: Development of flow control approaches to overload protect sensing cloud applications.** The developed flow control approaches address overload situations by adjusting the transmission rate. They aim to maintain the QoS by improving the timeliness on the expense of accuracy. Therefore, they are a complementary runtime mechanism to auto-scaling, which maintains the QoS on the expense of resource costs. The evaluation on the example of a sensing cloud application shows, that the approaches are able to maintain a QoS with varying overload intensities with a comparable performance to auto-scalers.

**C4: Design of rule-based coupling approaches to combine flow control with auto-scaling.** We present rule-based coupling approaches for coupling flow control approaches with auto-scaling. This allows to leverage the mechanisms of auto-scaling and flow control in managing sensing cloud applications. We evaluate, that a coupling is able to improve the QoS conformance.

**C5: Design of a sensing cloud application model for self-adaptive systems.** We present a sensing cloud application model for self-adaptive systems consisting of the cloud topology, effectors and probes. This model enables to design runtime management approaches and deploy them on sensing cloud applications in heterogeneous environments.

## 1.6. Outline

The thesis is structured as follows:

**Chapter 2.** This chapter introduces the foundations. It deals with the Cloud-IoT paradigm to introduce the domain of sensing cloud applications. We introduce the basic concepts of self-adaptive systems. We discuss auto-scaling and elasticity evaluation. We also introduce the basics of performance modeling of software system optimization.

**Chapter 3.** Chapter 3 introduces an analytical model for predicting performance metrics to scale microservices. It also introduces a simulation model that can be used to evaluate the performance of auto-scalers.

**Chapter 4.** This chapter introduces QoS cost functions for sensing cloud applications. These are suitable for measuring the impact of resource or transmission rate reconfigurations. Furthermore, an optimization framework is described, which provides interfaces to simulation engines and relevant concepts.

**Chapter 5.** This chapter discusses flow control approaches for sensing cloud applications, which adapt the transmission rate in overload situations to utilize the cloud service capacity without inducing time delays. The approaches are transferred and extended from existing congestion control procedures from TCP to IoT platforms. It presents an approach which utilizes request schemes which are well-known from TCP and introduces a capacity prediction module, which leverages the messaging paradigm of cloud services.

**Chapter 6.** This chapter introduces coupling mechanisms to combine auto-scaling and flow control approaches. The presented approaches are based on activation control and decide at runtime which strategies are activated.

**Chapter 7.** The SEIA Framework is introduced in this chapter, which is a framework for the prototypical development of runtime management approaches. It defines concepts of sensing cloud applications and can be instantiated on different cloud platforms. It allows to operate multiple strategies concurrently and supports smart device reconfigurations.

**Chapter 8.** Chapter 8 deals with the validation of the presented approaches by means of case study systems. These are used to cover the contributions in Chapter 3-6. One of the case study systems is derived from a Smart Heating solution by Bosch.

**Chapter 9.** This chapter surveys related work to the presented approaches. It discusses approaches from closely related fields.

**Chapter 10.** This chapter outlooks future work and concludes the thesis with a summary.

# 2. Foundations

This chapter introduces foundations that the following chapters build upon and elaborates how they relate to them. We introduce the context of this work in Section 2.1 by outlining the Cloud-IoT paradigm. In Section 2.2 we introduce qualities of sensing cloud applications. We summarize auto-scaling systems and elasticity evaluation in Section 2.3. Subsequently, Section 2.4 introduces TCP flow control concepts. In Section 2.5 we presents self-adaptive systems. Finally, Section 2.6 introduces optimization methods.

## 2.1. The Cloud-IoT Paradigm

This section introduces the context of this work by presenting the foundations of the Cloud-IoT paradigm. First, it presents the concepts of cloud computing and IoT. Then, it introduces the architectural integration and the microservice architectural style. Both are important for the evaluation of performance metrics in chapter 3 and the flow control approaches in chapter 5. Finally, we introduce qualities of sensing cloud applications, which the QoS cost functions presented in chapter 4 are based on.

### 2.1.1. Underlying Concepts

The Internet-of-Things (IoT) is a disruptive technology [28] which aims to provide advanced services by integrating physical and virtual things [57]. Based on the massive amount of data produced by billions of connected devices, the IoT is part of the `Big Data` problem, which is characterized by the practice of collecting and analyzing data sets with a high volume, velocity and variety [50]. Therefore, IoT applications require many resources to transport, process and persist the data. The data comprises of measured parameters of the physical world as well as changes of it.

Cloud computing offers a flexible infrastructure to provision resources on demand with the illusion of unlimited computation and storage capabilities. It allows customers to dynamically use scalable computing or storage resources with a pay per use cost model. The cloud environment is based on virtualization layers to enable a high resource utilization and scaleability. There are three different layer in cloud computing, which are *Infrastructure-as-a-Service (IaaS)*, *Platform-as-a-Service (PaaS)* and *Software-as-a-Service (SaaS)*. IaaS provides a virtual infrastructure, enabling developers to use computing and storage capabilities without requiring physical resources on their side. It mitigates the need for a data center by providing an environment to deal with virtual machines, storage or load balancers. PaaS enables developers to focus on higher levels of stack by providing a runtime environment for applications. Therefore a PaaS platform includes resources like operating systems or programming languages that automatically scales to meet the application demands. The PaaS is closely related to container platforms like Kubernetes, which orchestrate applications within runtime containers. SaaS is the most abstract layer and provides a multi-tenant architecture in which users access via the Internet a software provided by SaaS provider. Therefore it needs no installation of the service and can be accessed from anywhere. In the context of this thesis, we focus on operating cloud services on a PaaS.

The *Cloud-IoT paradigm* emerges from the combination of IoT and Cloud Computing. Whereas IoT devices are limited in terms of processing and storage capacity, cloud computing provides virtually unlimited storage and processing capabilities. Therefore, they are complementary to each other and are expected to provide benefits in specific application scenarios. According to [13] Cloud-IoT enables a range of paradigms which are centered around sensing the environment and providing a service. Applications enabled by Cloud-IoT which are expected to strongly impact everyday life are healthcare, smart home, smart cities and smart mobility. Healthcare applications allow to increase the quality of medical services, e.g. by collecting vital data. Smart home, cities and mobility interact strongly with the surrounding environment to enable a smart actuation with the environment or generate insights. The paradigm behind many of these applications is *Sensing-as-a-Service* and *Sensing-and-Actuation-as-a-Service*. Whereas the first aims to provide ubiquitous access to sensor data, the second aims to enable automatic control logic in the cloud. These paradigms impose several challenges, e.g. in terms of the sensing accuracy, timeliness or reliability. We refer to these applications as sensing cloud applications.

### 2.1.2. IoT Platforms

Figure 2.1 shows an IoT reference architecture as surveyed in [35].

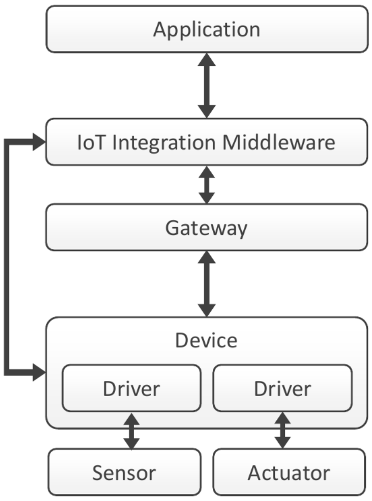


Figure 2.1.: IoT reference architecture [35].

It consists of devices, IoT Integration middelware, application and optionally a gateway. The devices are hardware components which are connected to sensors and actuators. Sensors are able to perceive the environment and measure parameters. The actuator is able to interact with the environment. It relies on drivers to access sensor and actuators. A device is connected to an application by an IoT integration middleware, which is also labeled as IoT platform. The middleware is responsible to manage connected devices. It is able to receive data from connected devices and provide it to applications. Additionally, it enable applications to send commands to be executed by the actuators. Gateways enable devices to overcome communication limitations, e.g. based on lacking a protocol, by forwarding communication or translating data. Many IoT platforms, such as ThingsBoard[1] and FIESTA-IoT platform[2], go beyond a mere gateway between the smart devices and cloud services by offering functionalities for data collection and device control, which could be leveraged at runtime.

### 2.1.3. Microservice Architectural Style

In order to leverage the capabilities of cloud computing in terms of scalability and maintainability the focus in software industry has shifted from monolithic architectures to the microservice architectural style [5]. Whereas traditional applications exhibit a monolithic architecture which tends to put multiple functionality into a single process the microservice architectural style separates functionalities into self-contained services. Breaking down software to loosely coupled and highly cohesive modules offers multiple benefits in terms of flexibility and evolvability [26]. By supporting scaling operations on a fine-granular level infrastructure costs can be reduced up to 70 % compared to a traditional monolithic architecture [72].

Microservices are typically connected to each other via a lightweight communication protocol, most commonly REST or message queues [30]. In this thesis we focus on the communication via message queues provided by a message broker system. There are many message broker systems which differ in their mechanism and feature set and a short survey describing the most popular message broker systems can be found in [44]. However, most process messages in a FIFO manner and provide a high degree of availability. The state of a message queue can be a challenge in such a system if the rate of incoming messages continuously exceeds the rate of outgoing messages eventually resulting in performance and reliability degradations.

## 2.2. Qualities of Sensing Cloud Applications

The business case of sensing cloud applications is to sense the environment by multitude of sensors in order to analyze it or actuate with it. Whereas some applications consume a stream of data others need to store the sensor data [46]. However, in both cases the data quality is critical. In this section, we introduce qualities of sensing cloud applications and use them as foundation for constructing QoS cost functions in chapter 4.

[1] https://thingsboard.io/

[2] http://fiesta-iot.eu

### 2.2.1. Terminology

In cloud environments a Service-Level Agreement (SLA) is a contract which defines a service based on the agreement between a provider and a customer [18]. Service-Level Objectives (SLO) are part of a SLA and denote measurable characteristics of the SLA such as availability or performance [62]. A SLO may be composed of one or more Quality-of-Service (QoS) measurements. These measurements can be combined to determine the achievement value, e.g. availability SLO may depend on multiple components, each of which may have a QoS availability measurement.

### 2.2.2. Data Qualities

Academia and industry have discussed six data stream qualities extensively: accuracy, completeness, timeliness, duplication, orderliness and consistency [74]:

- **Accuracy.** The accuracy describes the conformance of recorded value to actual value. Smart devices sense the environment and transmit the collected values to a processing cloud service. Decisions based on inaccurate data are likely to be faulty, therefore accuracy is an important quality for sensing cloud applications. The accuracy depends on the sensor quality, the sensing rate and the information loss through pre-processing on a smart device.

- **Completeness.** Completeness is defined in terms of any missing value. As a lack of accuracy, incomplete data can result in faulty decisions, thus impacting the data quality to a great extent.

- **Timeliness.** Timeliness requires that values are up-to-date. The timeliness is affected by the delay between transmitting and receiving sensor data. The processing delay can impact the decision making of sensing cloud applications, since the sensed environment can not be processed in time. The deployment decisions, network bottlenecks or underprovisioning of the cloud service can cause delays which results in timeliness violations.

- **Duplication.** Duplication refers to the existence of exactly the same records due to errors. This can impact the decision making.

- **Orderliness.** Orderliness demands that the data is collected in chronological order.

- **Consistency.** Consistency demands that a different value can only occur if there is more than one state. It also refers that the structure of related data remains the same.

Maintaining the qualities can be considered as responsibility of both transport and application layer. Modern message broker like RabbitMQ provide delivery and processing confirmation mechanisms which aim to maintain duplication, completeness and orderliness. Consistency is a quality which should be maintained by the application itself.

## 2.3. Elasticity in Cloud Computing

In this section, we introduce the foundations to auto-scaling and elasticity evaluation. Elasticity is defined as a property of cloud systems by representing the degree to which a system is able to adapt to workload changes by provisioning resources in an autonomous manner [39]. Auto-Scalers are a method which aims to make cloud systems elastic by autonomous resource provisioning based on performance metrics. The quality of auto-scalers can be evaluated using elasticity metrics. The evaluation of performance metrics introduced in chapter 3 and its validation in chapter 8.4 are build upon those.

### 2.3.1. Auto-Scaling

Auto-Scaling refers to a method to provision resources in an autonomous manner to maintain the QoS. In recent years, it experience a widespread usage to runtime manage cloud applications by leveraging their elasticity. There are many techniques to realize auto-scaling, e.g. based on control theory, threshold-based rules or machine learning as surveyed in [49] and [19]. More sophisticated approaches like [6] rely on workload forecasting, online resource demand estimation and performance models to improve the adaptation behavior. These techniques deeply rely on metrics which are able to represent the condition of the cloud system.

Auto-Scalers follow the MAPE-K loop by collecting information from a cloud environment, analyzing the information, planning scale operations and executing them [66]. The information usually contains a performance metric, which is used in decision making. Popular performance metrics are infrastructure metrics like the CPU utilization or the arrival rate. However, they may also be based on SLA violations or the response delay.

Threshold-based rules auto-scaling exhibits a widespread use in industry due to the simplicity and high availability among cloud providers like Amazon EC2. Rules in this context consist of a condition and an action to be executed. Usually they define a lower and upper threshold for a performance metric. If the current value of the metric exceeds a threshold, the auto-scaling systems scales application instances in or out.

Typically, they are classified into `reactive`, `proactive` or `hybrid`. A reactive auto-scaler analyzes the current state of the system to plan scaling decisions. A proactive auto-scalers analysis historical data to plan ahead. There are also hybrid approaches, which provide both, a reactive and a proactive component [7]. In general, auto-scalers aim to minimize the resource costs and at the same time maintain the QoS. In this work, we focus on reactive, threshold-based auto-scalers.

### 2.3.2. Elasticity Evaluation

In this section we introduce the elastic speedup measure proposed by SPEC RG Cloud [38] in order to quantify the elasticity achieved by auto-scalers. The measure quantifies the elasticity by reflecting the supplied and demanded resources within the measurement period regarding timing and accuracy aspects. Whereas the timing aspects are expressed by the share of time in an under- or overprovisioned state the accuracy describes the absolute deviation of each state in respect to the demanded resources. Both aspects are normalized over the measurement period and each aspect is aggregated to a single *accuracy* and *timeshare* metric using a custom weight for under- and overprovisioning. The elastic speedup measure is based on a speedup vector $s_k$ for a benchmarked platform $k$. The speedup vector $s_k$ is computed with the accuracy and timing aspects of the benchmarked platform $k$ and a baseline platform *base*:

$$s_k = (\frac{accuracy_{base}}{accuracy_k}, \frac{timeshare_{base}}{timeshare_k}) = (accuracy, timeshare)$$

The elastic speedup measure for a benchmarked platform $k$ is the geometric mean of its speedup vector $s_k$:

$$elasticspeedupmeasure = \sqrt{s_{k_{accuracy}} + s_{k_{timeshare}}}$$

## 2.4. Flow Control and Congestion Avoidance

In this section, we introduce flow control and congestion avoidance mechanisms of TCP which are the foundations of the flow control approaches for sensing cloud applications presented in chapter 5. Furthermore, we introduce a fairness measure which is used to validate the performance of a TCP-inspired flow control approach in chapter 8.9.

### 2.4.1. TCP Flow Control & Congestion Avoidance

The Transmission Control Protocol (TCP) is a transport layer protocol used for the majority of the Internet data traffic. It provides a reliable transmission of data between applications. TCP has two key concepts to enable a reliable transfer of data: flow control and congestion avoidance.

Flow Control limits the rate of a sender to ensure that it sends only as much data as the receiver can process. A receiver actively manages a receive window to state how many bytes it can buffer.

Congestion avoidance ensures that the network link is fairly shared across multiple senders and able to handle the load. One of its main components is an increase and decrease algorithm to adjust the transmission rate in respect to the state of the network. Each sender manages its own request scheme and receives a congestion feedback by observing acknowledgments or timeouts, which also indicate, if a network congestion has occurred. If a sender notices a congestion, it decreases the transmission rate. If there is no congestion, the sender increases it. Request schemes can consists of different additive and multiplicative increase/decrease combination [22]. Each combination exhibits different characteristics in terms of efficiency or fairness. Most TCP variants utilize an Additive Increase/Multiplicate Decrease (AIMD) scheme [2], which allows senders to fairly share a bottleneck.

### 2.4.2. Jains Fairness Index

In distributed systems, in which a set of resources is shared by a number of users, fairness is an important consideration. Especially for TCP as the dominant transport protocol on the Internet it is important to achieve fairness to avoid congestions. In this regard, a scheme is considered as fair, if the throughput of each user is at least as large as that of the others sharing a bottleneck. Jain's Fairness Index [43] allows to quantify the fairness of a scheme and bounds the index between 0 and 1, whereas 1 means absolute fairness. It enables to identify underutilized connections. Based on $n$ users and $x_i$ as the throughput of the $i$th connection the fairness measure $\zeta$ can be computed as follows:

$$\zeta(x_1, ..., x_n) = \frac{(\sum_{i=1}^{n} x_i)^2}{n * \sum_{i=1}^{n} x_i^2}$$

## 2.5. Self-Adaptive Systems

This section introduces the concept of self-adaptive systems and the MAPE-K autonomous loop. These are the foundations of the framework for runtime management framework for cloud applications presented in chapter 7.

Self-adaptive systems are able to adjust their behavior in response to their perception of the environment and the system itself [73]. Therefore, it is a system, which can handle changes in its environment, the system itself and its goals autonomously. Figure 2.2 illustrates the conceptual model of a self-adaptive system.

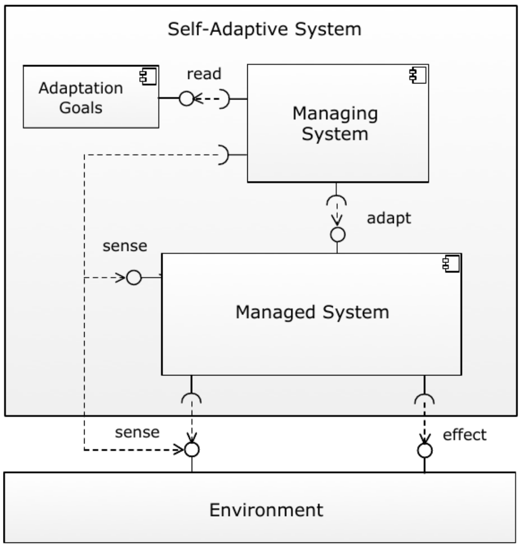


Figure 2.2.: Conceptual model of a self-adaptive system [73].

A self-adaptive system comprises of an environment, a managed system, adaptation goals and a managing system. The environment refers to the external world, which the self-adaptive system interacts with. The effects of the self-adaptive system on the environment are observed and evaluated. The environment can be sensed and effected through effectors and sensors. The managed system refers to application which realizes the systems domain functionality. In case of a sensing cloud application, sensing and processing sensor data is done by the managed system. The adaptation goals are the concerns of the managing system over the managed system and usually relate to software qualities. In case of cloud applications, it may be concerned with resource reconfiguration goals to maintain the performance of the managed system. The managing system manages the managed system and therefore utilizes adaption logic that is associated with the adaptation goals.

One of the most common structure to model managing systems is the MAPE-K control loop [41]. Conceptually it consists of a monitoring, analyzing, planning and execution phase which share a common knowledge base. The managed system is observed within the monitoring phase by sensor. Planned adaptations are executed by using the effectors of the managed system. Decisions are made based on the knowledge, which contains information on the system structure and state. Therefore, a managing system may utilize knowledge about the architecture of the managing systems, as proposed for the Rainbow framework [31].

## 2.6. Optimization

In this section, we briefly present Differential Evolution (DE) and Pareto optimality. The optimization framework for sensing cloud applications presented in chapter 4 is instantiated with DE as optimization method in section 8.2 of the validation. In section 8.5 we create Pareto curves in order to capture the characteristics of flow control approaches for sensing cloud applications.

### 2.6.1. Differential Evolution

Differential Evolution (DE) is a heuristic, multi-dimensional genetic optimization method [69]. It optimizes a problem by maintaining a population of candidate solutions which are combined to existing ones and filtered in terms of the best fitness. DE is an algorithm for black-box optimization and addresses problems with minimal knowledge about its structure [65].

Formally it aims to minimize a function $\chi : \mathbb{R}^n \to \mathbb{R}$. The function takes a candidate solution with $n$ dimensions as vector of real number $\mathbb{R}^n$ as argument and outputs a real number $\mathbb{R}$ expressing their fitness. Let a candidate solution be $x \in \mathbb{R}^n$. A solution $m$ is a global minimum if $f(m) \leq f(p)$ with $p$ as the overall search space.

The algorithm is based on an initial population size $p$, a crossover probability $CR \in [0, 1]$ and a differential weight $F \in [0, 2]$. The crossover probability expresses the probability of recombining each dimension. The differential weight impacts the magnitude of change.

Until a termination condition is reached, e.g. by a sufficient fitness, it does the following for each candidate solution $x$ in the population:

1. Pick three candidate solutions $a, b, c$, which are distinct to $x$.

2. Randomly pick a dimension $R \in 1, ..., n$ which will be recombined.

3. Then, compute a new candidate solution $y = [y_1, ..., y_n]$ by picking for each $i \in 1, ..., n$ a random number $r_i \in [0, 1]$.

4. If $r_i < CR$ or $i = R$ compute $y_i = a_i + F * (b_i - c_i)$. Else, do not recombine and set $y_i = x_i$.

5. If the candidate solution $y$ results in a better fitness such that $f(y) < f(x)$, replace $x$ with it.

The optimization performance is impacted to a great extent by the DE parameters $F$, $CR$ and $R$. However, it is well-studied with adaptation rules devised in [23] and [69].

### 2.6.2. Pareto Frontier

The Pareto efficiency describes when a set of parametrizations or allocations is optimal [17]. The optimality means, that there are no other alternative allocations which could improve a criterion without worsening another. In this regard, a Pareto frontier is a set of allocations that are Pareto efficient.

Figure 2.3 illustrates a Pareto frontier on the example of resource costs and QoS of a cloud application. Each allocation A, B, C and D represents a resource configuration, e.g. the provisioning of each microservice. That means, that there is no other resource configuration, which results in a higher QoS without increasing the resource costs. The Pareto frontier allows a service operator to make decisions based on the trade-offs.

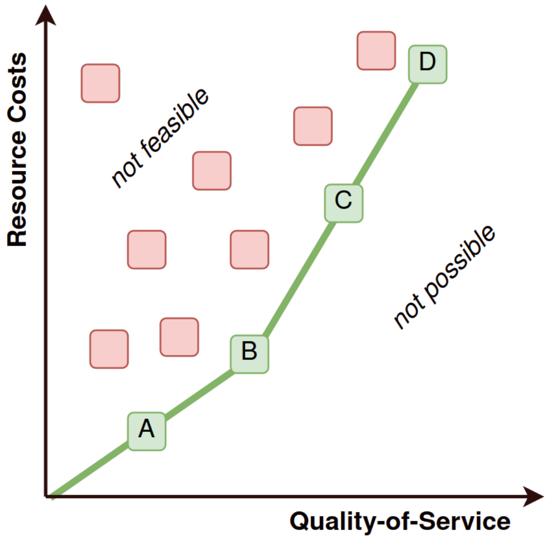


Figure 2.3.: Illustration of a Pareto frontier marked as green line. The green boxes represent allocations which are Pareto efficient. The red boxes represent Pareto inefficient allocations.

# 3. Evaluation of Performance Metrics for Scaling Decisions

This chapter introduces a microservice model, which is used to evaluate state of the practice auto-scalers relying on the CPU utilization for different microservice performance characteristics. Based on the widespread asychronous communication in microservice architectures via message queues, we extend the evaluation by investigating the suitability of message queue metrics.

Threshold-based rules auto-scaling is one of the most common techniques to scale resources in an autonomous manner. Scaling decisions are based on a threshold and recurring measurements of an application- or infrastructure-layer metric. The evaluation focuses on the CPU utilization of the microservice and performance and load metrics from the messaging middleware. Resources of a cloud infrastructure are prone to capacity variations in computation [24] or network [60][59] resulting in a changes of the service time. Since thresholds are configured in an ad hoc manner and are typically not adjusted during runtime, the thesis analyzes the effect of capacity and resource demand variations on the performance metrics:

**Research Question 1.** *How robust are infrastructure metrics for scaling microservice to capacity and resource demand variations?*

The research question is divided into the two sub questions, to investigate both: the impact of capacity or resource demand variations on the infrastructure metrics and its impact on the elasticity. The contribution are two-fold: first, by evaluating the quality of scaling decisions based on a popular performance metrics when faced with microservice characteristic variations, and second, by evaluating the suitability of queueing metrics. The results of this evaluation have been published in [33].

**Research Question 1.1.** *How are the infrastructure metrics affected by capacity or resource demand variations?*

**Research Question 1.2.** *How do capacity or resource demand variations affect the elasticity of threshold-based auto-scaling systems?*

## 3.1. Microservice Model

Sensing cloud applications receive and process sensor data using highly specialized microservices. The concern of the microservice can affect its resource characteristics to a great extent, e.g. a microservice which converts sensor data is more compute-intensive than a microservice storing data in a storage system. Figure 3.1 illustrates this on the example of a smart health application composed of microservices with varying resource characteristic communicating via messages queues.

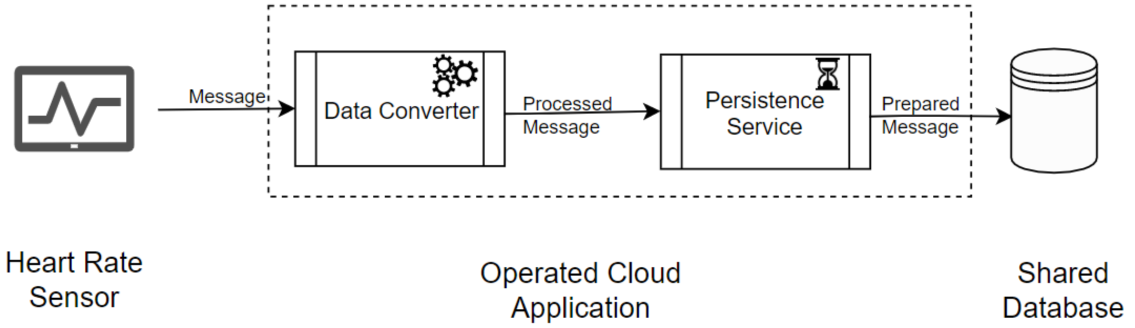


Figure 3.1.: A sensing cloud application consisting of a compute-intensive microservice (*data converter*) and a I/O-intensive microservice (*persistence service*).

We focus the evaluation on stateless microservices which are I/O- or CPU-bound. We use Kendall's notation [45] to model the microservice as a G/G/N stable queue, in which $N$ describes the number of service replicas. Each message $M$ is retrieved via a message queue before being processed. However, we neglect the impact of the communication protocol and model the resource demand for each $M$ as $D_M = (D_{CPU}, D_{IO})$. Using the microservices capacity $C = (C_{CPU}, C_{IO})$, we calculate the service time $ST$ for a message $M$ as follows:

$$ST(M, C) = D_{CPU_n} * C_{CPU} + D_{IO_m} * C_{IO_m}$$

## 3.2. Analytical model of infrastructure metrics

This section proposes an analytical model for a set of infrastructure metrics based on resource demand and capacity variations.

### 3.2.1. CPU Utilization

We model the CPU utilization as the ratio of time spent on the CPU to wait operations during serving a message $M$:

$$CPU(D_M, C) = \frac{D_{CPU} * C_{CPU}}{D_{CPU} * C_{CPU} + D_{IO} * C_{IO}}$$

Based on this model, the CPU utilization is strongly affected by variations in both, the computation and I/O capacity and by the resource demand of the message. The CPU utilization is not affected by changes in the overall service time, but by changes in the ratio of compute and I/O-operations.

### 3.2.2. Message Queue Metrics

Let $q$ be a queue, $l$ the number of messages in the queue, $p$ the production rate and $c$ the consumption rate. Let the service policy of $q$ be first-come, first-served (FCFS). This queue is illustrated in Figure 3.2.

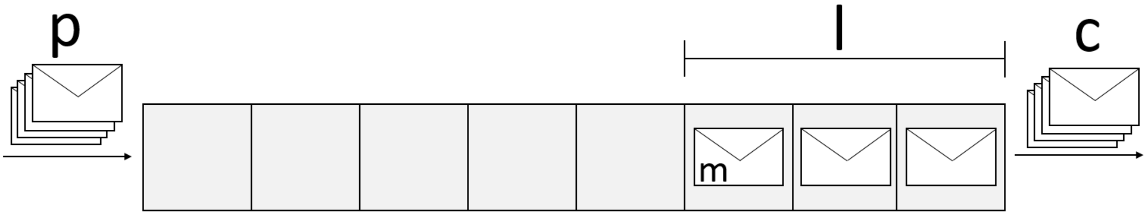


Figure 3.2.: Illustration of a message queue.

We model the consumption rate based on the service time $ST$ for a message $M$ of the microservice. Therefore, the consumption rate capacity $c_{max}$ depends on number of replicas $N_C$ of the cloud application:

$$c_{max}(t) = \frac{1}{N_C(t) * ST_M(t)}$$

We model the absolute growth of a message queue $\Delta l_q$ at a point of time $t$ as follows:

$$\Delta l_q(t) = \begin{cases} (p(t) * \Delta t + l(t)) - c_{max}(t) * \Delta t, & \text{if } c_{max}(t) * \Delta t \leq (p(t) * \Delta t + l(t)) \\ -1 * min((p(t) * \Delta t + l(t)), c_{max}(t) * \Delta t), & \text{otherwise} \end{cases}$$

The queue length $l(t)$ is therefore the integral over the queue growth:

$$l(t) = \int_0^t \Delta l_q(t)\,\mathrm{d}t$$

Based on this model, capacity and resource demand changes affect the consumption rate, which influences the queue growth, queue length and the queueing delay. Queueing metrics are affected by changes in the service time, but not by the ratio of compute and I/O-operations.

## 3.3. Simulation model

In this section, we propose a simulation models for cloud applications, in order to investigate the effect of resource demand and capacity variations on scaling decisions. Whereas the simulation model for cloud applications is based on a single service state of the art approach presented in [56], we extend it to simulate the influence of the environment on the service measurements and the reconfiguration delay of scaling decisions. Furthermore, we extend the simulation model to include smart devices as controlled components, as suggested in [64] in order to represent sensing cloud applications.

### 3.3.1. Model for Cloud Applications

In this section, we propose a simulation model to investigate the effect of resource demand and capacity variations on a threshold, as shown in figure 3.3. We derive the queueing model from [56]. We refine it by adding measurement providers to reflect different monitoring policies. We solve the service time and the CPU utilization of the queueing model analytically based on the models proposed in section 3.2. We denote $k \in \mathbb{N}$ slotted time, in which the arrival rate and decisions of the auto-scaler are counted. Furthermore, we model a scaling delay $T_{Scaling}$ for provisioning microservice replicas.

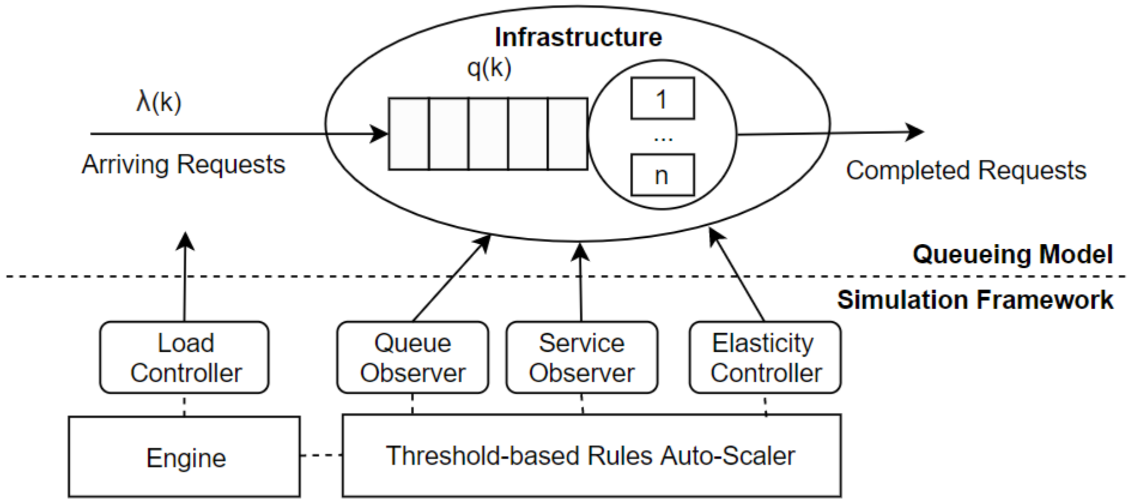


Figure 3.3.: Queueing Model for the cloud application with the probes and effectors of the auto-scaling system.

The simulation framework provides a CPU utilization which is calculated based on the I/O- and computation-time mix of processing messages.

The messaging middleware is an entity provided by the simulation framework. We provide a FIFO queue, which resembles the messaging paradigm of the AMQP, implemented by many popular message brokers. The rate of outgoing messages is determined by the processing rate of the queueing model and the rate of ingoing messages is determined by the load intensity. The state of the message queue is updated at each simulation step $k$.

Scaling decision are based on measured metrics of the infrastructure. Since monitoring is associated with costs, the measurements are typically repeated within an interval, which is large enough to capture changes with a specific accuracy. Furthermore, the measurements are processed with a specific policy, e.g. moving average to smooth them. To include these in the simulation, we model a measurement provider $MP$ as a tuple of an update interval $T_X$ and a moving average of size $N$.

### 3.3.2. Model for Sensing Cloud Applications

Figure 3.4 illustrates the extension of the simulation model to represent sensing cloud applications.

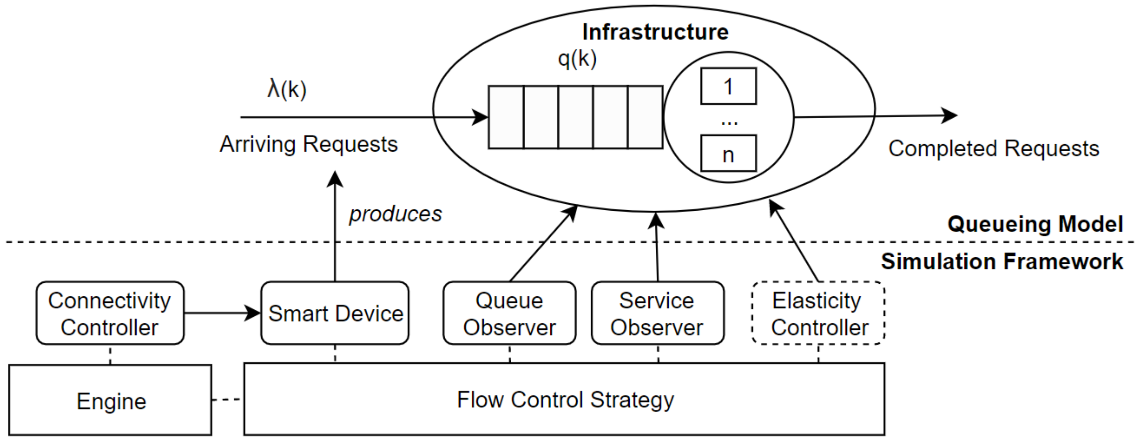


Figure 3.4.: Queueing Model for sensing IoT applications with smart devices as controlled components.

We introduce smart devices which are associated with a single data stream. The transmission rate of the smart devices contributes to the overall arrival rate for each simulation time step $k$. Smart devices include a latency distribution, in order to simulate the network based latency due to the distributed setup. Reconfiguring decisions are delayed by the latency at minimum and only done after transmitting the next sensor data if the information exchange policy is piggybacking. In order to adjust the number of connected devices, we introduce a connectivity controller. The number of connected devices is based on the experimental setup. The simulation model can be used to evaluate auto-scalers or flow control strategies in a sensing cloud application context.

## 3.4. Discussion

In this chapter we proposed an analytical model to describe the impact of resource demand and capacity variations on the infrastructure metrics. Furthermore we introduced a simulation model to predict auto-scaling decisions for a time-varying workload. The analytical model shows, that variations in the workload mix affect the CPU utilization greatly, whereas message queue metrics are not affect. However, message queue metrics are affected by the overall service time, whereas the CPU utilization is unaffected. The presented simulation model is able to simulate single-service cloud applications in a typical IoT setup, consisting of smart devices, a messaging middleware and the cloud solution. The simulation framework provides a concept of measurement providers, which degrade measurements according to an aggregation policy, usually a moving average. Furthermore it provides controllable smart devices and a connectivity controller to simulate sensing IoT scenarios. The simulation model can be used to evaluate auto-scalers but can be extended to evaluate any strategy, which affects the load or capacity of the cloud application, e.g.

auto-scaling or flow control of smart devices. Due to its simplicity its not able to simulate multi-service cloud applications or complex workflows and assumes homogeneous service instances.

# 4. QoS Cost Optimization of Sensing Cloud Applications

IoT platforms enable the Internet-of-Things by providing a platform, which allows smart devices to connect with cloud services [35]. Many IoT platforms go beyond a mere gateway between the smart devices and cloud services by offering functionalities for data collection and device control. This enables a runtime management of sensing cloud applications, which is able to dynamically reconfigure the cloud service by resource provisioning and the smart devices by adjusting the collection strategy, which determines the rate of which sensor messages are collected. Since the resolution and timeliness of sensor data affects the data quality, which is in turn important for analyzing the environment and actuating with it, it is important to find a trade-off in resource provisioning and collection rate in order to maximize the applications QoS. Therefore, we introduce QoS cost functions, which capture the impact of resource and message rate configurations. Additionally, we present an optimization framework to maximize the QoS conformance of sensing cloud applications by searching for an optimal configuration of the resources or runtime management approaches. Therefore, we address the following research question:

**Research Question 2.** *How to model the QoS conformance in terms of the impact of the resource configuration and the transmission rate of a sensing cloud application to enable an efficient resource management?*

The research question is divided into two sub questions:

**Research Question 2.1.** *How to model the QoS costs of a sensing cloud application?*

**Research Question 2.2.** *What is a sufficiently fast approach for optimizing the QoS conformance of a sensing cloud application?*

## 4.1. Application Model

Let a sensing cloud application consists of $N$ smart devices, an IoT platform and a cloud service consisting of $M$ microservices. Let the microservices be managed by an elastic cloud platform, which enables to scale each service horizontally on demand. Let the smart devices be managed by the IoT platform, which allows the smart devices to connect and to adjust the collection behavior. Let each smart device sense an environment $E_M$ and collect environment data with an internal sensing rate. Let the collected data be subsequently processed on-device to reduce the data volume and overhead for data transfer and management [46]. Based on the applications need, the on-device processing policy may combine or filter sensed data. Whereas `combining` describes the combination of multiple sensor values to a single value, e.g. by aggregation, `filtering` is about discarding sensor values, e.g. to capture the latest value. Let the IoT platform be able to configure the rate of which pre-processed sensor data is collected. The collection may take place by being requested from the smart devices by the IoT platform or by being pushed to the IoT platform by the smart devices. Let the IoT platform dispatch collected data to the cloud application via a messaging middleware. Figure 4.1 illustrates the application model.

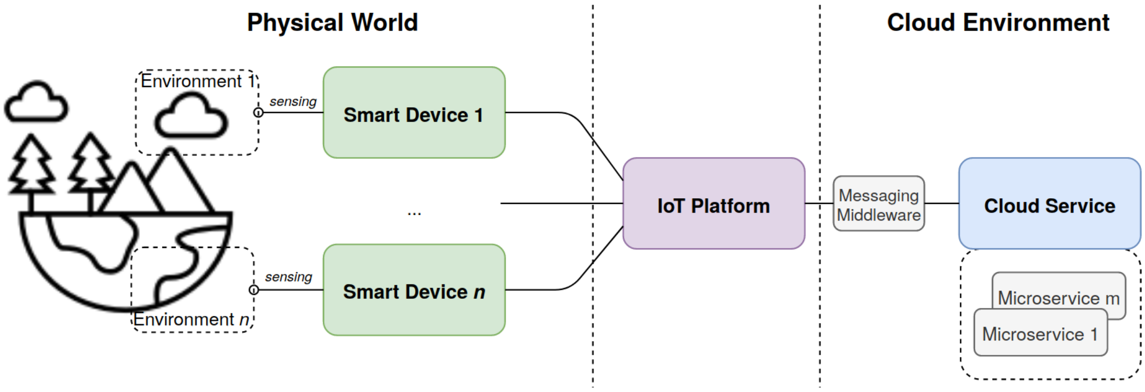


Figure 4.1.: Application model of a sensing cloud application consisting of smart devices, an IoT platform and a cloud service.

## 4.2. Qualities of a Sensing Cloud Application

In this section we select a subset of qualities introduced in chapter 2.2 which are affected by the resource provisioning of the cloud service and the message rate of the smart devices. In the following, we reason, why we deem each quality as relevant:

- **Timeliness.** The message rate configuration of smart devices collectively induce the load on the cloud service. The resource configuration of the cloud service affects its capacity. If the load exceeds the capacity the messaging middleware eventually accumulates messages, hence inducing processing delays. Therefore the timeliness is affected by an interplay of both configurations.

- **Accuracy.** The message rate configuration of smart devices affects the sensing rate of the environment. The time series of sensor data represent the state of the environment. If it does not capture changes in the environment it degrades the accuracy. Therefore, the accuracy is affected by the message rate. We also refer to the accuracy as *sensed accuracy*.

- **Resource Costs.** Based on the cost model of cloud environments the resource configuration determines the resource costs.

Qualities of data stream consider timeliness and accuracy as separated dimensions. The separation of the dimensions holds true for applications which aim to first collect everything and analyse it at a later point of time. However, on the viewpoint of time-critical cloud services, e.g. infrastructure management, the timeliness does affect the accuracy. Such cloud services use the received data to actuate with the environment or to enable decisions, therefore delayed data may impact the accuracy of its environmental model. In order to capture the interplay of timeliness and accuracy on the QoS of a sensing cloud application, we extend it with *perceived accuracy*, which is illustrated in figure 4.2:

- **Perceived Accuracy.** The perceived accuracy is a function of the accuracy of measured values at the time of arrival at the cloud application and the state of the environment, i.e. it becomes minimal if there is no processing delay and no measurement inaccuracy.

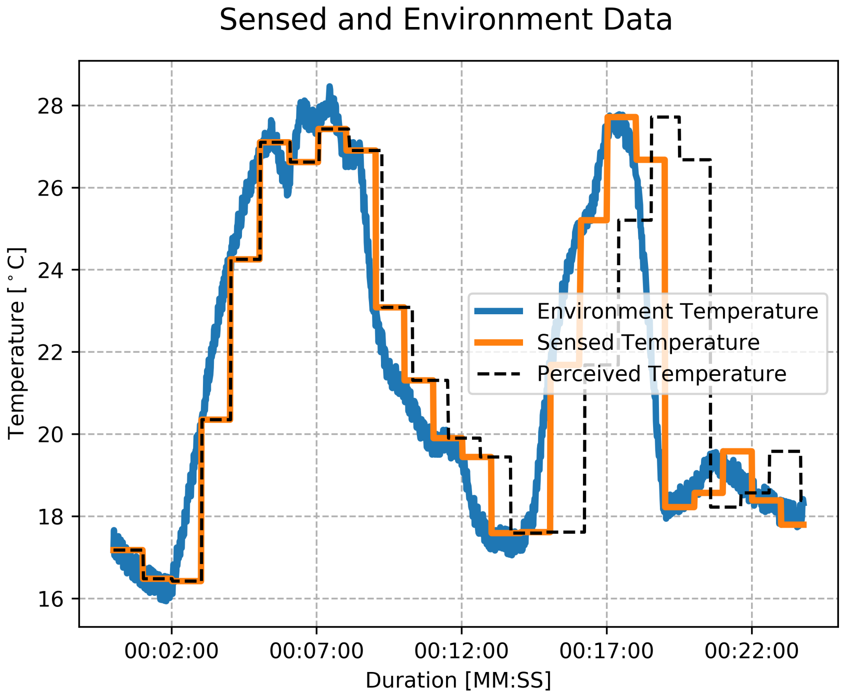


Figure 4.2.: Sensed, perceived and environment data on the example of temperature. The sensed accuracy is 4.67 %, whereas the perceived accuracy by the cloud service is 10.46 %, which is degraded because the data is delayed.

We do not consider duplication, consistency and orderliness as affected qualities by the resource and message rate configuration. Instead, we consider duplication and orderliness as responsibilities of the messaging middleware and consistency as responsibility of the application layer. We also consider completeness as the responsibility of modern message brokers since they support delivery and processing confirmation mechanisms.

## 4.3. QoS Metrics

This section introduces for each quality QoS metrics, which are used as input for QoS cost functions to express the conformance to the quality. Furthermore, we discuss, how they can be obtained in design- and runtime:

- **Timeliness.** In design- and runtime, the delay can be obtained by measuring the end-to-end latency of transmitted sensor data. Based on the rich monitoring capabilities of modern message brokers, the queueing delay can be selected as a proxy metric. Relying on the queueing delay has the benefit that is able to reflect an imbalance of the arrival and processing rate of messages without including delays by the communication link. This can be a valuable feedback for a runtime management system, which reconfigures the provisioning of a cloud service or the transmission rate of smart devices.

- **Accuracy.** The accuracy of sensed data can be expressed as the average measurement error. A sensing cloud application may demand a target accuracy expressed as a percentage of error, based on the sensed smart device time series and the actual environment model. Let the function to compute the achieved accuracy for a smart device $n$ be $acc_{sensed}(n)$ for a time span $\delta T$. In order to obtain the accuracy $acc_{sensed}$ it should be differentiated between design- and runtime. In design-time, the accuracy can be exactly expressed by relying on an environment model for each smart device in order to quantify the sensed accuracy. In runtime, a quantification can be done heuristically by determining the change between the current and the last collected value, as in [67]. Since each smart device senses a different part of the environment, the resulting accuracy differs for each. Therefore, we propose the average sensed accuracy $acc$ as an input metric for an accuracy cost function:

  $$acc = \frac{1}{N} \sum_{n=0}^{N} acc_{sensed}(smartdevice_n)$$

  The SLA of a sensing cloud application may impose a specific collection interval $collect$ on smart devices instead of demanding a target accuracy. The interval can be based on considerations of dynamics in the environment and the average interval, which is necessary to achieve a demanded accuracy. The collection interval is then set for all smart devices. Quantifying accuracy costs based on collection interval degradations may not be suitable in many cases, since it is unable to reflect the actual sensed accuracy, which depends on the changes in the environment in-between the interval. However, by being obtainable at runtime and declared as a target in SLAs it is a feasible input metric.

- **Perceived Accuracy.** The perceived accuracy describes a viewpoint of an application, in which it assumes that received sensor data represents the current environment state. Therefore the perceived accuracy is based on the interplay of the accuracy and timeliness of sensed data and demands a trade-off between the two dimensions. It can be obtained in design-time by comparing the environment state over the time with the state perceived by the cloud service. Let the accuracy be declared as $perc$.

- **Resource Costs.** Based on the pay-as-you-go cost model of many cloud platforms, resource cost functions require the amount of provisioned resources over the time to quantify the costs. By assuming homogeneous resource costs for each provisioned microservice, we provide the overall amount of provisioned resources $res$ as an input metric for a resource cost function:

$$res_{is} = \frac{1}{\delta T} \sum_{n=0}^{m} instances(microservice_n)$$

## 4.4. QoS Cost Functions

In order to quantify the qualities demanded by sensing cloud applications, we discuss accuracy, timeliness and resource cost functions. Furthermore we introduce a perceived accuracy cost function, which quantifies the accuracy based on the interplay of accuracy and timeliness. The cost functions enable a search for an optimal resource and message rate configuration.

### 4.4.1. Quality Cost Functions

Based on the multitude of applications with different quality demands, the concrete quantification of the costs should be done by a service operator. In the following we discuss for each quality how a service operator could set up the cost functions. The following cost functions assumes but are not limited to a target and worse value, as an input to a dedicated error function. The error functions are not defined here, but should be carefully selected by the service operator:

- **Timeliness.** SLAs may impose an acceptable delay $delay_{worst}$, often expressed as a percentile. This value can be compared to the processing delay $delay$ in order to quantify the costs of timeliness degradations. Additionally, the acceptable delay may be motivated by availability demands based on the SLA, which may require the data to be available within a certain frame of time. Let a timeliness cost function $Q_{Timeliness}$ quantify the resource costs based on the difference of $delay$ and $delay_{worst}$, e.g. by mapping the absolute or relative deviation to costs.

- **Accuracy.** Based on the use case of a sensing cloud application it demands a specific accuracy of sensing the environment, e.g. 10 %, which is hereby declared as target accuracy $acc_{target}$. Let $Q^{'}_{Accuracy}$ be a cost function, which quantifies the QoS costs based on the measured accuracy $acc$ and the target accuracy $acc_{target}$. For SLAs with a target collection interval $collect_{target}$, we propose an accuracy cost function $Q^{''}_{Accuracy}$, which is based on $collect_{target}$ and $collect$.

- **Perceived Accuracy.** Let $Q_{Perceived}$ be a cost function, which quantifies the costs based on the perceived accuracy $perc$ and a target accuracy $perc_{target}$.
- **Resource Costs.** Let the acceptable amount of provisioned resources be $res_{worst}$. Let the resource cost function $Q_{Resources}$ rely on $res_{worst}$ and $res$ in order to quantify the costs. Note, that $res$ is a non zero value, since a cloud service requires at least one active resource in order to be available. The cost function can include cost models based on the cloud provider, since many provide transparent pricing per leasing hour.

### 4.4.2. QoS Cost Function Sets

We propose two different sets of QoS functions, which can be selected based on the preference of the cloud service operator. The first QoS cost function set $\chi_1$ considers accuracy and timeliness as separate dimensions. Thus, emphasizing the ability to reconstruct the environment based on the sensed values with a specific accuracy. In this regard, timeliness only affects the delay at which data is available at the processing cloud application. Therefore, we construct the QoS cost function set $\chi_1$ with $Q_{Accuracy} \in \{Q^{'}_{Accuracy}, Q^{''}_{Accuracy}\}$ as follows:

$$\chi_1 = Q_{Accuracy} + Q_{Timeliness} + Q_{Resources}$$

The second QoS cost function set $\chi_2$ includes the perceived accuracy, which unifies timeliness and accuracy to a single dimension to express the accuracy of data received by processing application:

$$\chi_2 = Q_{Perceived} + Q_{Resources}$$

## 4.5. Optimization Goals

In this section, we briefly introduce optimization goals. The first optimization goal aims to optimize the resource configuration of the cloud service and the message rate of the smart devices to minimize the QoS. The second aims to optimize the configuration and selection of runtime management approaches which adjust the message rate and resource configuration.

### 4.5.1. Resource and Message Rate Configuration

Let the message processing capacity of a cloud service be a function of the resource configuration $P$, which holds the number of provisioned resources per (micro-)service. Let the transmission rate configuration $T$ describe the transmission rate of each smart devices. By transmission rate, we mean either the send rate of smart devices to the IoT platform or the collection rate of messages by the IoT platform.

Let each of the QoS cost function sets $\chi_1$ and $\chi_2$ contain the objectives, which are to be minimized. What is an optimal mapping $C : PxT \rightarrow \Re^{\chi}$ for a transmission rate configuration $T$ and a resource configuration $P$ such that the fitness of $\chi \in \{\chi_1, \chi_2\}$ is minimized collectively in weighting?

### 4.5.2. Runtime Management Configuration

Let the runtime management $R$ of a sensing cloud application consists of $N$ runtime management approaches. Let each runtime management approach affect a cloud application by reconfiguring the transmission rate of smart devices and/or the resource configuration of microservices. Let $R$ consists of $n$ parameters, with $n$ as the sum of parameters over $N$ approaches. The optimization problem is to find a candidate solution, which minimizes the QoS costs: $\chi : \mathbb{R}^n \to \mathbb{R}$ with $\chi \in \{\chi_1, \chi_2\}$.

## 4.6. Optimization Framework

In this section we introduce a framework for optimizing the configuration or runtime management of sensing cloud applications. Figure 4.3 illustrates the components of the framework.

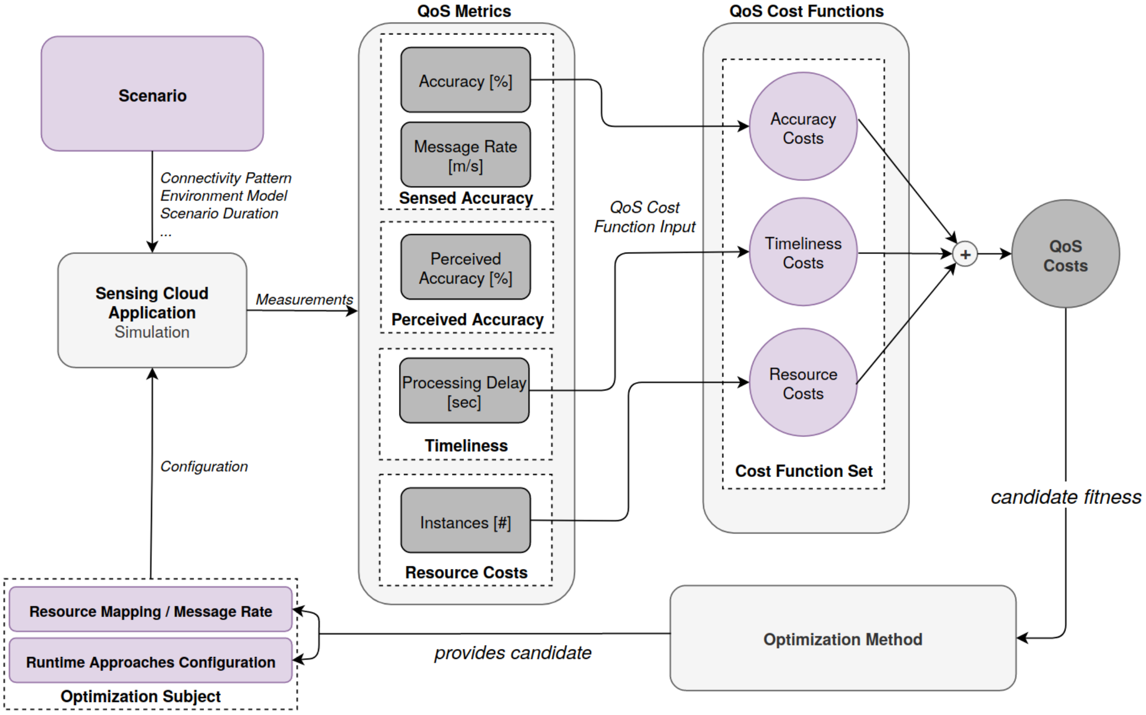


Figure 4.3.: Framework for optimizing the QoS costs of a sensing cloud application. A service operator has to provide a specific scenario and a set of QoS cos functions, which maps QoS input metrics to costs. An optimization method evaluates the cumulative QoS costs and provides new candidate solutions.

A service operator has to provide a set of QoS cost functions, which are able to quantify the QoS metrics to costs. The QoS cost functions are based on the demanded qualities of the sensing cloud application and constructed based on the trade-off of these. The cumulative QoS costs are used as a feedback for an optimization method. Based on the achieved costs, an optimization method provides candidates. Since the optimization goals are multi-dimensional and the systems non-trivial, we assume that it is not feasible to solve the optimization problem analytically. Instead, to improve the efficiency, a heuristical optimization method should be selected. The candidates provided by the optimization method aim to select configuration parameters as candidate solutions, which minimize the cumulative QoS costs. The optimization process demands a simulation engine, which is able to simulate cloud applications in conformance to the application model presented in this chapter. Additionally it has to support a dynamic reconfiguration of the resources in order to enable the simulation of runtime management approaches. The service operator has to provide a scenario, which contains the connectivity pattern of smart devices, scenario duration and provides, if applicable, environmental models, which are sensed by the smart devices. Based on these inputs, the simulation engine performs measurements to provide the QoS input metrics, which have been introduced in this chapter. They are subsequently used as an input for the provided QoS cost functions. A service operator can introduce a stopping criterion, if a target QoS cost, iteration depth or duration is reached.

## 4.7. Discussion

In this chapter we have identified qualities for sensing cloud applications which are affected by the transmission rate and resource configuration. Based on the set of qualities and the complex interplay of transmission rate and resource configuration we consider the search for cost optimal configurations as a multi-dimensional optimization problem. We discussed design- and runtime input metrics for the QoS cost functions, which enable to quantify the QoS costs of a specific dimension based on cost functions. Additionally, we presented an optimization framework, which can be instantiated to search for optimal resource configurations. The resource cost function assumes, that the resources of the microservices are equally expensive. This may not hold true on a practical perspective, since many cloud providers, e.g. Amazon AWS, offer different pricing in respect to the machine type. The presented set of QoS cost functions assume that the smart devices perceive the environment with one sensor, whereas many IoT applications require the input of multiple sensors on the smart devices. Additionally, the presented approach does not consider energy consumption, which is an important quality since many IoT devices are battery constrained. An extension of this work can consider battery power of IoT smart devices as an additional trade-off to accuracy and timeliness. Overall, we deem the QoS cost functions and the presented approach as essential to optimize the QoS of sensing cloud applications. We instantiate the presented framework and cost functions in the validation chapter.

# 5. Time-driven Flow Control of Smart Devices

This chapter presents flow control approaches which adjust smart devices at runtime in order to improve the QoS conformance. We transfer congestion control mechanisms of transport layer and refine it to be applicable for sensing cloud applications and take advantage of their architectural design decisions. The presented flow control approaches are time-driven by aiming to avoid congestion-induced delays and to utilize the available cloud services capacity. We discuss the integration of these approaches into sensing cloud applications. In contrast to a static message rate and resource configuration as discussed in chapter 4 runtime approaches are able to cope with performance uncertainties due to capacity and resource demand variations. We address the following research question:

**Research Question 3.** *What are suitable flow control approaches to improve the QoS conformance of sensing cloud applications?*

The chapter is structured as follows: first, we clarify the concepts by emphasizing on the differences to transport layer flow control. Then, we discuss the architectural integration into an IoT application. Finally, approaches are presented and concluded in a discussion.

## 5.1. Underlying Concepts

In this section, we introduce flow control for sensing cloud applications. The term flow control is derived from distributed systems and a well-known concept of TCP. Therefore, we first briefly recapitulate flow control in TCP and then discuss how we distinguish its concepts to the flow control of sensing cloud applications. Figure 5.1 illustrates the relation of both to each other.

Flow control in TCP ensures, that a sender is not overwhelmed by a receiver by sending more packages than it can consume. A receiver actively manages a receive window to state how many bytes it can buffer. In contrast, congestion avoidance ensure that the network link is fairly shared across multiple senders and able to handle the load by adjusting the transmission rate based on a congestion feedback. In many TCP implementations a sender assumes a network congestion, if messages could not be delivered. The transmission rate of a sender is influenced by both mechanisms.

In sensing cloud applications smart devices sense the environment and transmit messages which are processed by a cloud service. The capacity of a cloud service emerges from the performance characteristics of its components. Therefore, the capacity may

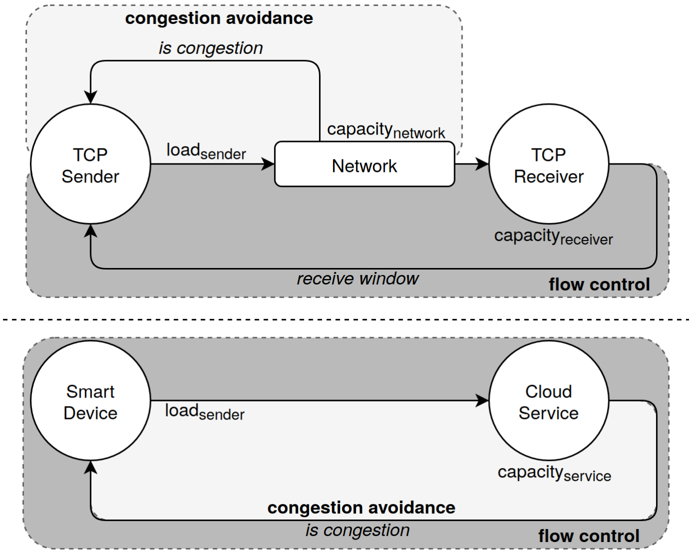


Figure 5.1.: Illustration of flow control and congestion avoidance in TCP and for sensing cloud applications. The flow control of sensing cloud applications utilizes end-to-end congestion avoidance techniques to ensure, that a cloud service is not overwhelmed by smart devices.

be unknown or subject to variations. The smart devices of a sensing cloud application transmit sensor data with a specific rate which is based on accuracy considerations. The overall load can exceed the capacity of the cloud service which degrades the timeliness by inducing congestions. We focus on avoiding overloading the cloud application with messages and exclude network limitations.

Based on the conceptual differences the terminology of TCP congestion avoidance and flow control does not result in an exact fit. Instead, we introduce them as flow control for sensing cloud applications which utilizes congestion avoidance mechanisms to satisfy QoS goals. Applying congestion avoidance mechanisms to sensing cloud applications enables to maintain the timeliness demand of the cloud service by adjusting the transmission rate of smart devices. By adjusting the transmission rate to a value which utilizes the capacity of a cloud service to a high degree, they maintain the timeliness but eventually degrade the accuracy. Since they prioritize timeliness over accuracy, we classify them as time-driven flow control approaches. We distinguish between two closely related modes:

- **Congestion-Avoidance Mode.** The goal is to maximize the accuracy without timeliness degradations. Therefore, the flow control aims to fully utilizes the capacity of the cloud service without inducing congestions.

- **Overload-Protection Mode.** The goal is to overload protect a sensing cloud application by maintaining the timeliness on the expense of accuracy when faced with resource constraints. If the load does not exceed the capacity, the flow control utilizes the capacity to a degree which satisfies the accuracy demands.

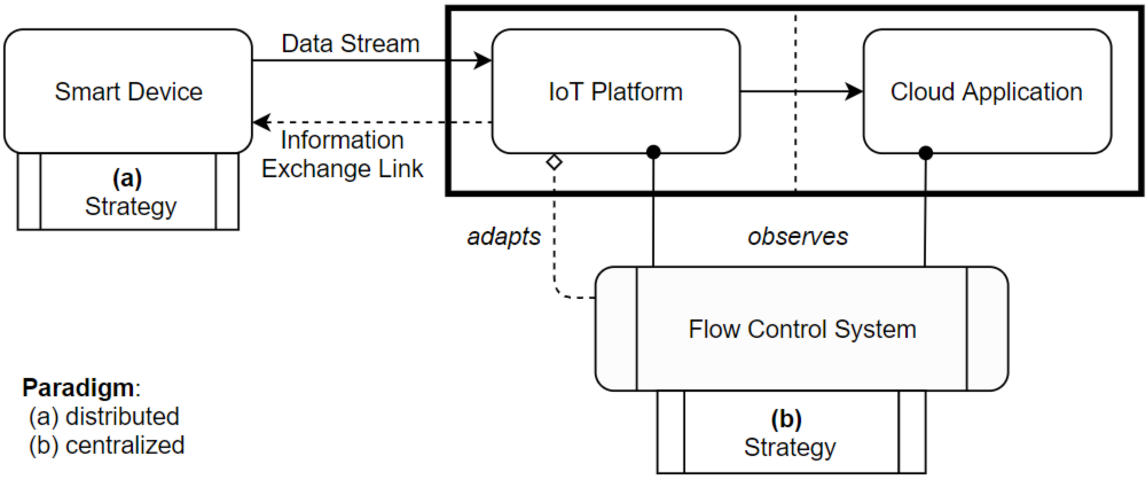


Figure 5.2.: Architectural integration of the flow control system. (a) distributed: smart devices utilize a strategy which relies on information provided by the flow control system. (b) centralized: smart device reconfigurations are based on decisions of a global strategy.

## 5.2. Integration into Sensing Cloud Applications

In this section, we discuss the integration of time-driven flow control approaches into a sensing cloud application.

### 5.2.1. Architectural Integration

Figure 5.2 illustrates the architectural integration of a flow control system to an IoT platform. The IoT platform manages connected devices and provides received sensor data to corresponding cloud applications. Therefore, it acts as a communication link to devices, which provides probing and adaption interfaces to a flow control system. A flow control strategy can be deployed in a distributed or centralized manner. In a distributed setup, the flow control system provides feedback to the adaptation engines of the smart devices in order to let them adjust the data stream. In a centralized setup, the flow control system determines and provides the adjustments to the smart devices. The flow control system observes the cloud application and the IoT platform, in order to obtain measurement data used to recognize congestions and reconfiguration decisions.

### 5.2.2. Information Exchange Mechanism

Information exchange between smart device and the flow control system should leverage the communication link of the IoT platform to smart devices. Since many IoT platforms offer a REST-API for smart devices to connect and to receive data, piggybacking can be realized by enriching the HTTP response with information. Piggybacking aims to reduce the amount of communication overhead required for transmitting information to smart devices by utilizing the acknowledgment of received sensor data to inject the feedback.

However, the information exchange flow is degraded, if a smart device stops or slow downs transmitting. IoT platforms utilizing MQTT could deploy control channels. Control channels may induce a higher network overhead but are not affected by the interval between requests.

### 5.2.3. Congestion Observer

A congestion observer is an integral element of time-driven flow control since the transmission rate adjustments are based on a binary congestion feedback. We define a congestion observer $O$ as a tuple of a metric $M$, a threshold $T$ and a window size $N$, such that $O = (M, T, N)$.

#### 5.2.3.1. Metric Selection

A congestion occurs if the cloud application's capacity can not cope with the load thus inducing a processing delay which degrades the QoS. Based on the cloud applications topology, there are multiple possibilities to use a measured metric as a proxy for a congestion. A popular metric to reflect the services' utilization is the CPU. Another possibility could be a package loss, e.g. by message discarding, or a high response time. Since we focus on sensing cloud applications, which retrieve sensor data of a message queue provided by the IoT platform, we propose to use message queue metrics. Since messages are processed in a FCFS manner, an imbalance of the production and consumption rate results eventually in a non-zero queue length or queueing delay. Furthermore it excludes delays due to network congestions, which are out of the operational scope of the time-driven flow control approaches.

#### 5.2.3.2. Configuration

By providing the congestion feedback to the flow control strategies, the congestion observer impacts the adaptation behavior to a great extent. Whereas the congestion-avoiding flow control strategies may not require any application-specific knowledge, an application-aware configuration of the congestion observer can improve the quality of those. Therefore, a queue length or queueing delay threshold should be selected based on the SLA, to avoid a too eager or too sluggish observer.

We propose, to configure the adaptation interval of flow control strategies in respect to the reactivity of the congestion observer. The reason is that a flow control strategy can consider the effects of previous adaptations before deciding to adjust the transmission rate. An adaptation interval that is too short to capture the effect of a reconfiguration decisions may results in unnecessary adaptations which degrade the adaptation quality. If the adaptation interval is too large, the flow control system reacts sluggish. Therefore the configuration should consider the behavior of the congestion observer and the time it needs to adjust smart devices.

### 5.2.4. Transmission Rate Boundaries

The accuracy sensed by smart devices emerges from environmental changes and the resolution of sensor data. By sensing the environment cloud applications demand a specific accuracy. Flow control approaches may enforce boundaries to control the degree to which a transmission rate can be adjusted. The boundaries should be based on implicit or explicit accuracy considerations, such that a transmission rate adjustment is capped by an upper boundary and a lower boundary. In practice, a SLO may state a transmission interval required for the cloud service. This can be transferred as upper boundary, since a shorter interval does not contribute to the SLO fulfillment. If a SLO state a target accuracy, the flow control approach may enforce an upper boundary for the transmission rate, at which the target accuracy is met. We deem a lower boundary as optional. It may limit the congestion avoidance efficiency by capping the degree to which the load can be affected. However, we propose to set the lower boundary to a non-zero value, if the flow control information exchange relies on piggybacking, in order to maintain the control flow to the smart devices.

## 5.3. TCP-Inspired Flow Control

In this section, we introduce the TCP-inspired flow control for sensing cloud applications which is based on the congestion avoidance mechanisms of TCP. The results have been published in [32].

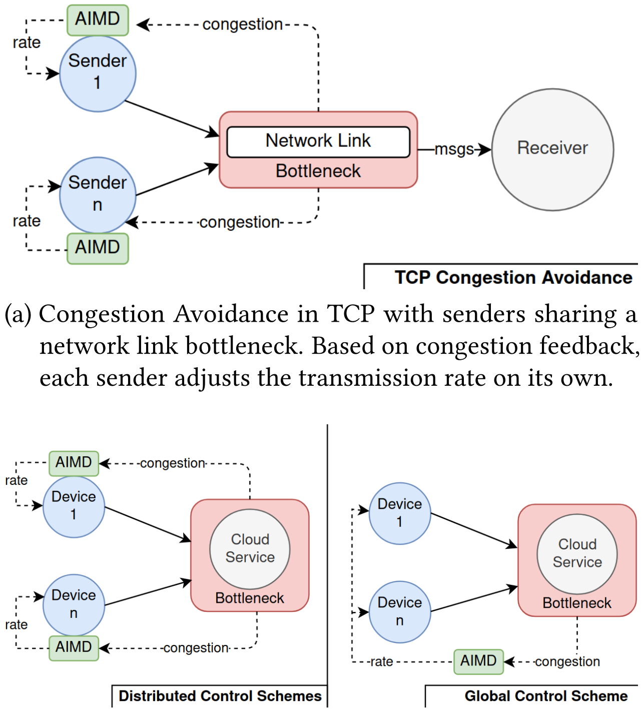
### 5.3.1. Conceptual Differences

In the following, we describe conceptual differences to the TCP congestion avoidance mechanisms.

Figure 5.3a illustrates the congestion avoidance in TCP. Each sender utilizes an increase and decrease algorithm to adjust the transmission rate based on a binary congestion feedback. Most TCP variants utilize an Additive Increase/Multiplicate Decrease (AIMD) scheme [2], which allows senders to fairly share a bottleneck.

In contrast, the flow control of sensing cloud applications considers the capacity of the cloud service as the bottleneck. Therefore, all smart devices share the same bottleneck. As illustrated in figure 5.3b, this allows to perform adaptations with two paradigms: distributed and global. Distributed request schemes are similar to TCP, since each smart device utilizes its own request scheme and adjust its transmission rate based on the congestion feedback. A global request scheme allows to adjust the transmission rate of all smart devices, such that adjustments are inherently fair. Therefore, fairness remains only a criteria for a distributed operation. In order to recognize congestions, we propose to rely on the congestion observers described in section 5.2.3 of this chapter and to adapt the transmission rate $T$ after a specific interval $\tau_{Adaptation}$. Based on the conceptual difference to TCP in terms of the adaptation paradigm and its fairness implications we propose to extend the request schemes by the following candidates:

- **AIMD.** Additive increase and multiplicative decrease, with $a > 0$ and $0 < b < 1$:



(a) Congestion Avoidance in TCP with senders sharing a network link bottleneck. Based on congestion feedback, each sender adjusts the transmission rate on its own.

(b) Adaptation paradigms in the TCP-inspired flow control for sensing cloud applications. In a distributed mode each smart devices utilizes a request scheme on its own. In a global mode a request scheme is used to adapt all smart devices.

Figure 5.3.: Illustration of the conceptual differences of congestion avoidance in TCP and for the time-driven flow control of sensing cloud applications. The request scheme is represented by AIMD.

$$T(t+1) = \begin{cases} T(t) + a, & \text{if no congestion} \\ T(t) * b, & \text{if congestion} \end{cases}$$

- **AIAD.** Additive increase and additive decrease, with $a > 0$ and $b < 0$:

$$T(t+1) = \begin{cases} T(t) + a, & \text{if no congestion} \\ T(t) + b, & \text{if congestion} \end{cases}$$

- **MIAD.** Multiplicative increase and additive decrease, with $a > 1$ and $b < 0$:

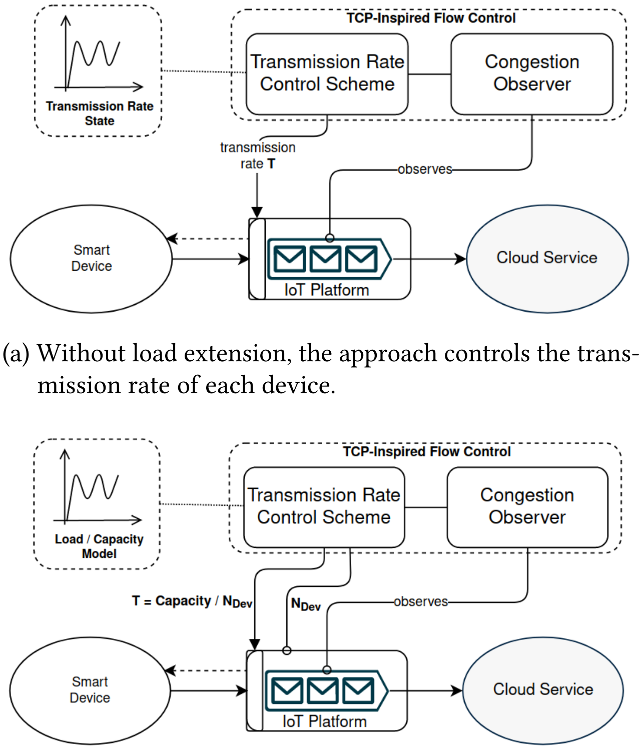
$$T(t+1) = \begin{cases} T(t) * a, & \text{if no congestion} \\ T(t) + b, & \text{if congestion} \end{cases}$$

- **MIMD.** Multiplicative increase and multiplicative decrease, with $a > 1$ and $0 < b < 1$.

$$T(t+1) = \begin{cases} T(t) * a, & \text{if no congestion} \\ T(t) * b, & \text{if congestion} \end{cases}$$

### 5.3.2. Load Model Extension

Figure 5.4a illustrates the approach without load extension. The request scheme controls the transmission rate of each device. Therefore, the impact of an adaptation step affects the overall load on the cloud service in dependency to the number of active devices. In order to maintain a proportionality of adjustments, such that each change affects the load independently to the number of connected devices, we propose to control a load model $i$ in the global request scheme instead of the transmission rate $T$. Then, we calculate the transmission rate of each smart device with $T(t+1) = \frac{i(t)}{N_D}$. Figure 5.4b illustrates the load extension, in which the the control scheme holds the current load on the cloud service and shares it fairly across $N_{dev}$ connected devices. This results in the need to obtain the number of connected devices, e.g. from IoT platform, in order to use it in adaptation decisions.



(a) Without load extension, the approach controls the transmission rate of each device.

(b) With load extension, the approach controls the overall load and shares it fairly across all devices.

Figure 5.4.: Illustration of the TCP-inspired flow control with and without load extension.

### 5.3.3. Overload Protection Mode

In the following we describe how to integrate boundaries into the TCP-inspired flow control. As discussed in section 5.2.4 the reason is to not utilize the cloud capacity to a greater extent as demanded by accuracy considerations. Therefore, the transmission rate adaptations are capped by a minimal transmission rate $T_{lower}$ and a maximal transmission rate $T_{upper}$. In between, the adaptations are based on the request scheme. We refine the request scheme adaptations as follows:

$$T(t+1) = max(min(T(t+1), T_{upper}), T_{lower})$$

### 5.3.4. Discussion

In this section, we presented a transfer of congestion avoidance mechanisms of TCP to flow control smart devices. It adapts the transmission rate of sensor data streams of smart devices, to achieve a high throughput and low processing delay. We have proposed a global and distributed setup of which a global setup enables to use request schemes which are not fair in a distributed setup. We proposed a load model extension, which aims to make the change by adaptations independently to the number of connect devices. We expect, that the the load model extension does not behave differently in a static connectivity scenario. However, in scenarios, in which the number of devices varies, we assume, that the extension results in a more consistent adaptation behavior. We deem the congestion avoidance mechanisms of TCP as suitable to cope with the challenge of maintaining the QoS of sensing cloud applications. An advantage of this approach is, that it is able to cope with a varying processing rate or a changing number of connected devices without relying on a performance model. However, we recommend to rely on application knowledge to properly configure the approach in terms of congestion recognition and request scheme parameters. The underlying increase/decrease algorithms have been shown to result in a high resource utilization and low delays. Overall, we expect this to translate to a high timeliness conformance but accuracy degradations in overload scenarios. This can be a beneficial addition to runtime manage time-driven sensing cloud applications.

## 5.4. Capacity-Estimating Flow Control

In this section, we present a congestion avoidance approach, which estimates the cloud applications' capacity at runtime. It aims to improve the TCP-inspired congestion avoidance by using an estimation to adjust the transmission rate instead of probing the capacity. We assume, that it results in a speed up in coping with dynamics, e.g. variations in the capacity or the number of connected devices. The results have been published in [34].

### 5.4.1. Capacity Estimation

Many sensing cloud applications retrieve sensor data of a message queue. The departure rate of messages is limited by the consuming cloud applications capacity. In overload situations, which eventually result in a congestion, the cloud capacity is insufficient to cope with the rate of arriving messages. This result in a non-zero queue length.

We introduce a moving average $M_{est}$ of size $N$, which is updated with measured rate of outgoing messages. However, it is only updated in overload situations, in which the measured queue length exceeds a threshold $T_{QLength}$. This threshold has to exceed the estimated capacity rate $C_{est}$, such that $T_{QLength} \geq C_{est}\Delta t$. This aims to avoid to degrade the estimated capacity, if the total number of remaining messages is not enough to fully utilize the cloud applications' capacity.

The service operator has to provide an initial capacity estimation $C_{initial}$ in order to let the approach work.

### 5.4.2. Transmission Rate Calculation

Based on the assumption that each sensor data demands the same amount of resources on the cloud application, we predict the currently supported transmission rate $T'_{estimation}$ of each device at a time $t$ using the cloud application's estimated processing rate $C_{CloudApplication}$ and the number of connected devices $N_{Devices}$ such that:

$$T'_{estimation}(t) := \frac{C_{est}(t)}{N_{Devices}(t)}$$

### 5.4.3. Phases

The congestion avoidance approach consists of two phases: a *congestion avoidance phase* and a *congestion protection phase*, as illustrated in figure 5.5.

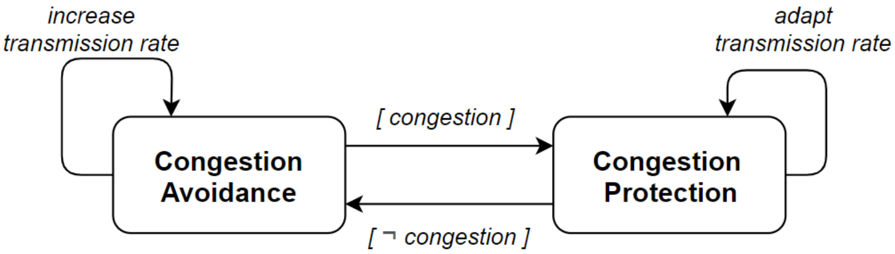


Figure 5.5.: Illustration of the Capacity-Estimating Congestion Avoidance phases.

**Congestion Protection.** During the congestion protection phase, the goal is to mitigate or resolve an overload situation. This is done by calculating a suitable transmission rate based on the number of connected devices and the estimated cloud capacity, such that the application is fully utilized. Since a congestion is characterized by a non-zero message queue, it induces a message processing delay. In order to let the message queue recover, we reduce the applications' utilization by a factor $k_{protection} < 1$, therefore ensuring that the queue size decreases over time.

$$T'_{protect}(t) := T'_{estimation}(t) \cdot k_{protection}$$

If the congestion has been resolved, the approach transits to the `Congestion Avoidance` phase.

**Congestion Avoidance.** During a congestion, the cloud applications' capacity is estimated. If there is no congestion, the estimation may not reflect the current capacity of the cloud application, since it is based on previous measurements. A decreased capacity eventually result in a congestion, which refines the capacity estimation. However, with an increased capacity, the estimation will not be refined, which results in an underutilized service. Therefore, we probe the current capacity by introducing a factor $k_{avoidance} > 1$ to increase the load on the application. We adapt the transmission rate in each adaptation interval $\tau_{Adaptation}$:

$$T'_{recover}(t) := T'_{estimation}(t - \tau_{Adaptation}) \cdot k_{avoidance}$$

If this results in a congestion, it transits to the `Congestion Protection` phase, in order to refine the capacity and resolve the congestion.

### 5.4.4. Overload Protection Mode

We refine the phases as illustrated in figure 5.6. Now, they have a different semantic, since the the `Congestion Avoidance` phase is replaced by the `Congestion Recovery` phase, which tries to restore an adjusted transmission rate to the provided upper boundary.

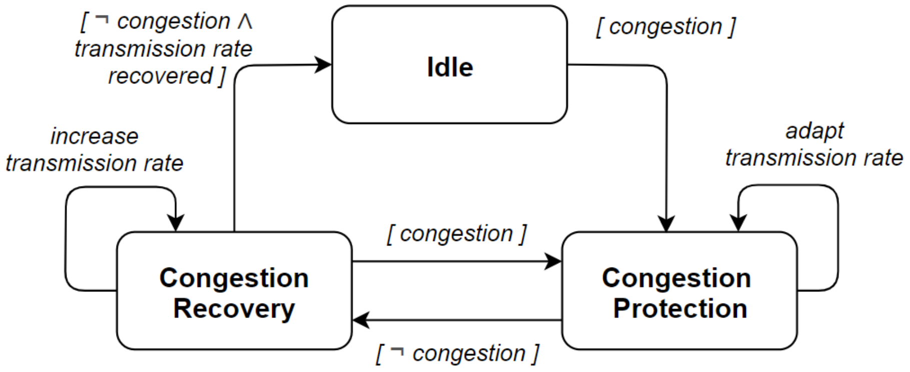


Figure 5.6.: Illustration of the capacity-estimating congestion avoidance refined and extended phases.

**Idle.** During the idle phase, the flow control system does not make any operation since the transmission rate is at the saturation point and there is no congestion.

**Congestion Protection.** During a congestion, we enforce the lower transmission rate limit as follows:

$$T'_{protect}(t) := max(T'_{cap}(t) \cdot k_{protect}, T_{lower})$$

If the congestion has been resolved, the approach transits to the `Congestion Recovery` phase.

**Congestion Recovery.** During the congestion recovery phase, the flow control systems aims to restore the transmission rate to the original value. Therefore, we limit the transmission rate adaptation as follows:

$$T'_{recover}(t) := min(T'_{recover}(t-T) \cdot k_{recover}, T_{upper})$$

If this results in a congestion, it transits to the `Congestion Protection` phase, in order to refine the capacity and resolve the congestion. If the transmission rate is restored and it has not result in a congestion, it transit to the `Idle` phase.

### 5.4.5. Discussion

The presented approach supports cloud applications, which receive sensor data by a message queue, to create a capacity estimation of the cloud application during runtime. This induces an architectural constraint which can be mitigated, if an online resource demand estimating approach is used, e.g. LibReDe [68].

The approach aims to improve the TCP-inspired congestion avoidance by introducing a runtime capacity estimation. By continuously probing the available capacity by inducing non-severe congestions, it results in an oscillating transmission rate. However, we assume that the capacity estimation contributes to a higher QoS conformance by reducing the convergence time to the capacity.

## 5.5. Discussion

We presented in this chapter flow control approaches, which aim to fully utilize the cloud applications' capacity with a low message processing delay, to improve the QoS conformance. Whereas the first approach is a TCP-inspired congestion-avoiding flow control, the second refines it by estimating the cloud applications' capacity at runtime. The presented approaches can be operated in an overload protection mode, in which they utilize an upper boundary for the transmission rate based on accuracy considerations. In congestion avoidance mode, they enable to utilize the capacity of the cloud service without inducing congestions, therefore affecting timeliness and accuracy. The presented approaches require application knowledge, which should be used to configure the congestion observer, the adaptation interval and their parameters. Overall, they aim to contribute to the QoS conformance by improving timeliness and degrading accuracy. Therefore, we expect them to complement the runtime management mechanism of auto-scaling, which improves timeliness on the expense of resources.

# 6. Coupling Mechanisms for Runtime Strategies

Cloud applications are usually operated with auto-scaling systems, which aim for provisioning resources to cope with the current resource demand. Therefore, they improve timeliness on the expense of resource costs. In contrast, the time-driven flow control approaches presented in chapter 5 adapts the transmission rate of devices to improve timeliness on the expense of accuracy. Using auto-scaling or flow control in isolation may result in a strong degradation of the QoS conformance. We aim to improve the overall QoS conformance by coupling both types of runtime management. We address the following Research Question:

**Research Question 4.** *What are suitable coupling mechanisms to improve the QoS conformance of sensing cloud applications by combining flow control with auto-scaling?*

We propose rule-based activation control mechanisms, which enable to set each strategy as active or inactive. This enables coupling approaches to be agnostic to the internals of the strategies. Based on activation rules, the mechanisms decide which strategy to enable, aiming to improve the overall QoS conformance. We introduce the following coupling mechanisms:

- **Concurrent Coupling**: A mechanism to deploy multiple runtime strategies on a service without any coordination.
- **Threshold-based Rules Coupling**: A mechanism, which decides based on a threshold-based rule set, which strategy to activate. We propose rule sets tied to the qualities of sensing cloud applications.
- **Fuzzy Rules-based Coupling**: A mechanism, which utilizes fuzzy rules to decide, which strategy to activate.

## 6.1. Coupling Strategy Metamodel

A coupling strategy requires effectors to control the life cycle of the coupled strategies. Based on its goals, it decides at runtime, which strategy to activate. Therefore, a coupling strategy may not be involved with the internals of the coupled strategies. Figure 6.1 shows the metamodel of coupling strategies.

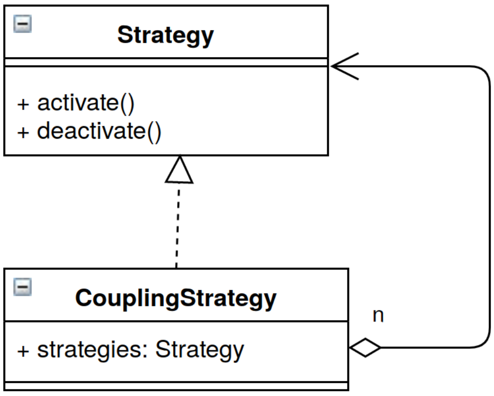


Figure 6.1.: Metamodel of a coupling strategy.

Since a coupled strategy contains strategies, it can also contain instances of itself, allowing, e.g., to enable or disable multiple provisioning strategies at once.

## 6.2. Strategy Classes

In this section we select strategy classes which reconfigure the transmission rate of smart devices or the resource configuration of cloud services at runtime. We consider the following as feasible candidates for a coupled runtime management:

- **Time-driven Flow Control (Congestion Avoidance).** Introduced in chapter 5, it aims to fully utilize the available capacity by adjusting the transmission rate. Therefore it enables to balance timeliness and accuracy.

- **Time-driven Flow Control (Overload Protection).** This class aims to minimize processing delays in overload situations by transmission rate adjustments. Therefore it aims to improve the timeliness of a sensing cloud application on the expense of accuracy. It caps the load on the cloud service in non-congestion scenarios based on accuracy considerations.

- **Accuracy-driven Flow Control.** An IoT data collection presented in [67], which adjust the transmission frequency based on measured changes in the environment in order to maintain a target accuracy.

- **Auto-Scaling.** By (de-)provisioning resources to meet the resource demand, auto-scalers aim to maintain the timeliness on the expense of resource costs. It reduces resources costs in non-underprovisioned scenarios.

Table 6.1 provides an overview of the strategies and affected qualities.

| Strategy | Capacity | Load | Accuracy | Timeliness | Resources |
|---|---|---|---|---|---|
| Time-driven Flow Control (Congestion Avoidance) |  | o | o | o |  |
| Time-driven Flow Control (Overload Protection) |  | o | o | o |  |
| Accuracy-driven Flow Control |  | o | o |  |  |
| Auto-Scaling | o |  |  | o | o |

Table 6.1.: Runtime strategy mechanisms affecting the load or capacity of the cloud service and their aimed improvement on the QoS costs.

## 6.3. Concurrent Coupling

Let a concurrent coupling activates each coupled strategy at every point of time. Therefore, this mechanisms does not involve any decision making. Instead, the contribution to the QoS conformance is based on the emerging behavior of the coupled strategies. This may result in the need to tune the parameters of each for a given application scenario. In the following, we discuss conceptual synergies and challenges of coupling auto-scaling with the presented flow control strategies:

- **Time-driven Flow Control (Congestion Avoidance).** A coupling with resource provisioning and congestion-avoiding flow control is conceptually not feasible, since this class aims to fully utilize the available capacity. Since auto-scalers usually operate on a load metric, they are triggered by the high service utilization, resulting in a windup of provisioned resources.

- **Time-driven Flow Control (Overload Protection).** This class of strategies, aims to reduce the timeliness costs by adjusting the transmission rate and thus affecting the accuracy in situations, in which the timeliness is violated. Whereas it adjust the load, resource provisioning approaches adjust the capacity. Therefore, both aim to improve the timeliness but on the expense of different qualities. Therefore, they are able to harmonize by sharing the expense of timeliness on accuracy and resource costs. However, in order to avoid a domination of one approach over the other, they need to be carefully configured and selected under consideration of the application scenario. Furthermore, they may rely on the same metric to identify congestions or service utilization, e.g. the message queue length, which emphasize the need to carefully tune their efficiency.

- **Accuracy-driven Flow Control.** This class of strategies aims to adjust the transmission rate in accordance to the current accuracy demands of the application considering changes in the measurements. Therefore, it results in a high frequency, if there are many changes in the environment and in a low frequency, if there are only minor changes. It can be coupled with resource provisioning, since the current capacity demand is determined by the required accuracy.

Overall, concurrent coupling is a straightforward mechanism to combine two or more strategies with each other. Note, that it has not to be managed by a global runtime management system but can be the consequence of deploying two or more runtime management approaches which are not aware of each other. The limitations of a concurrent coupling are mainly based on the non-existent coordination of strategies. Therefore, we assume, that a higher degree of QoS conformance can be reached, by coordinating strategies based on QoS metrics.

## 6.4. Rule-Based Coupling

In the following section we present threshold-based rule sets, which activate strategies based on boolean expressions. We introduce three different rule sets, motivated by maintaining different qualities.

### 6.4.1. Metamodel

Figure 6.2 shows the meta model of the rules-based coupling strategy.

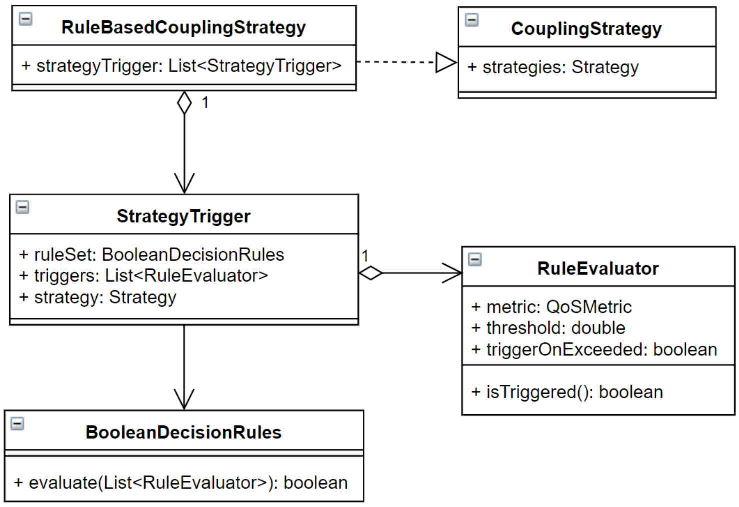

RuleBasedCouplingStrategy
+ strategyTrigger: List<StrategyTrigger>
CouplingStrategy
+ strategies: Strategy
1
StrategyTrigger
+ ruleSet: BooleanDecisionRules
+ triggers: List<RuleEvaluator>
+ strategy: Strategy
1
RuleEvaluator
+ metric: QoSMetric
+ threshold: double
+ triggerOnExceeded: boolean
+ isTriggered(): boolean
BooleanDecisionRules
+ evaluate(List<RuleEvaluator>): boolean

Figure 6.2.: Metamodel of a the rule-based coupling strategy.

**StrategyTrigger** For each managed strategy, the rule-based coupling strategy utilizes a strategy trigger in order to decide, when to activate it. The strategy trigger itself utilizes a set of binary feedback rule evaluators, which are evaluated altogether using a provided boolean expression.

**RuleEvaluator** The rule evaluator consist of a provided threshold and a metric. It allows to evaluate, if the value exceeds the threshold.

### 6.4.2. Accuracy-driven Overload Protection

**Goal.** State of the practice runtime management utilizes auto-scalers to cope with load variations. However, if faced with overload situations, auto-scalers may not react fast enough, such that the system experience periods of underprovisioning resulting in timeliness violations. In practice, auto-scalers are limited in the number of provisionable resources based on economical or resource constraints. Furthermore, cloud service may face bottlenecks, which are out of the operational scope of the service provider, which threaten both, the timeliness and the availability. For this reason, we propose a coupling, which aims to maintain a high accuracy and only degrades it in severe overload situations as a fallback option.

**Rule Set.** Based on threshold-based violations of the timeliness and accuracy QoS cost function, we active the auto-scaler(s) or the flow control as follows:

$$activate_{auto-scalers} = true$$

$$activate_{flow-control} = accuracy \lor timeliness$$

The auto-scaling system is always activated, which is expressed by a rule, which evaluates as true. The overload protection flow control is activated, if the timeliness has exceeded a threshold, or the accuracy is degraded. Whereas the timeliness trigger allows to reduce the timeliness violations by the overload protection mechanism of the flow control strategy, the accuracy trigger allows the transmission rate to be recovered, when the overload situation has been resolved.

Timeliness and accuracy metrics may be obtained by utilizing QoS cost functions or by relying on proxy metrics, e.g. queue length or queueing delay as timeliness and an absolute deviation of the transmission rate to the demanded value as accuracy. In either case, the threshold has to be selected carefully in order to enable a fallback mechanism, which does not eagerly degrade the accuracy.

**Discussion.** The presented rule set allows to deploy a runtime management system, which does not interfere with the usual auto-scaling but complements it with an overload protection flow control mechanism. It provides a fallback mechanism to maintain the timeliness and availability of a cloud service in severe overload situations. Since it activates auto-scaling all the time, it does not need to interact with these systems, such that the coupling strategy is transparent for existing auto-scalers.

### 6.4.3. Cost-driven Overload Protection

**Goal.** In contrast to the accuracy driven overload protection, this coupling strategy aims to reduce resource costs by degrading the accuracy in overload situations.

**Rule Set.** Based on threshold-based violations of the timeliness and accuracy QoS cost function, we active the auto-scaler(s) or the flow control as follows:

$$activate_{auto-scalers} = accuracy \lor resources$$

$$activate_{flow-control} = true$$

The overload protection flow control is always activated, which is expressed by a rule, which evaluates as true. Auto-scaling is activated, if the accuracy degradation has exceeded a provided threshold. However, the auto-scaler(s) are also activated, if resources induce costs, since it indicates, that the overload situations has not been resolved or the auto-scalers have not deprovisioned resources.

Resource costs can be obtained using a QoS cost functions or by relying on proxy metrics, e.g. the number of provisioned resources compared to a target provisioning.

**Discussion.** The presented coupling allows to use auto-scaling systems in severe overload situations, in which a further accuracy degradation is more expensive than provisioning resources. It needs to interact with the auto-scaling approaches, but not with the overload protecting flow control. For this reason, the strategy is transparent to the flow control systems.

### 6.4.4. QoS-based Coupling

**Goal.** The QoS-based coupling strategy aims to improve the QoS conformance by utilizing a set of QoS cost functions for each strategy. Therefore, the provided thresholds for each QoS cost function decides how the coupling system behaves and allow, based on their granularity, a fine tuning for specific application scenarios.

**Rule Set.** Each strategy $i$ relies on the following rule set:

$$activate_{strategy_i} = accuracy_{T_i^1} \lor timeliness_{T_i^2} \lor resources_{T_i^3}$$

Each strategy is activated, if any cost functions evaluates as true, when the cost value exceeds the provided threshold $T_i^x$.

**Discussion.** The QoS-based coupling allows to couple any strategy, based on activation rules on the QoS cost functions. In contrast to the previous presented approaches, it allows to define complex rules, which are not constrained by letting a strategy be activated all the time. However, based on the number of thresholds, it is complex to configure the threshold properly and should therefore be subject to a search approach.

### 6.4.5. Discussion

In this section, we have presented a metamodel of rule-based coupling strategies, which utilize a boolean algebra evaluation engine and a rule set in order to (de-)activate strategies on runtime based on QoS cost functions. Therefore, the presented mechanism is able to observe the effect of activated strategies and use it for coupling decisions.

## 6.5. Fuzzy Rules-Based Coupling

In this section, we present a fuzzy rules-based coupling, which aims to improve the QoS conformance by applying a set of rules. In contrast to the rules-based coupling, which relies on boolean logic, the fuzzy rules assign truth values between 0 and 1 and enable to hold a partial truth. Furthermore, it relies on set of rules, which fuzzify the QoS cost functions presented in chapter 4. Therefore, this approach aims to provide a mechanism for coupling mechanisms based on a set of reusable rules. The presented coupling mechanism aims to combine auto-scaling with overload protection flow control using a set of provided rules and QoS cost functions.

### 6.5.1. Fuzzification

Let a service operator provide a QoS cost function set consisting of normalized cost functions in the range of 0 to 1. For each QoS cost function, we propose a fuzzification based on three states: `good`, `ok` and `bad`. Figure 6.3 shows the fuzzy logic.

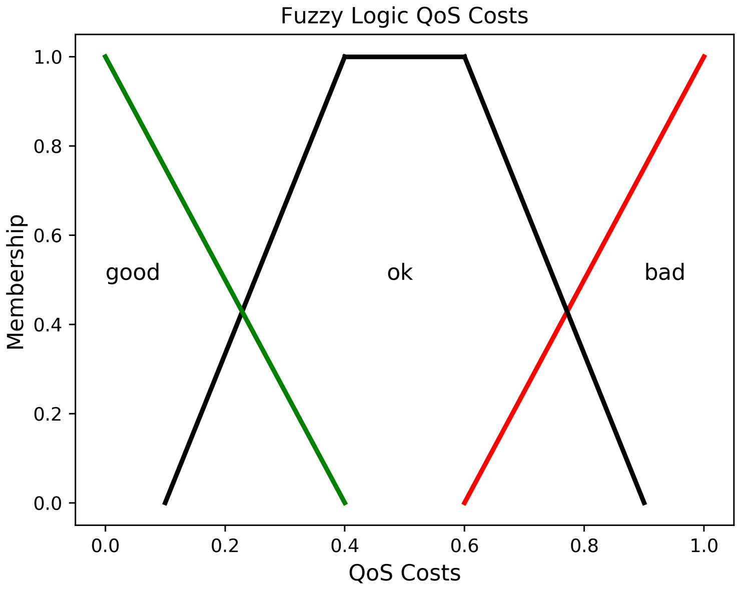


Figure 6.3.: Fuzzy logic for the output of a given QoS cost function.

Let `timeliness`, `accuracy` and `resource costs` be input variables. Based on these variables, we aim to express a set of reusable rules. Note, that the QoS cost functions needs to be linearly in this case, in order to maintain the linguistic semantics. Therefore, QoS cost functions needs to be constructed in a manner, which reflects the linguistic categorization.

### 6.5.2. Defuzzification

Figure 6.4 illustrates the defuzzification in order to decide, which strategy to activate.

It either operates flow control, auto-scaling or both simultaneously. The states have a logical continuity in order to allow a logical transition between the membership functions. The $x$-value is determined by the center of gravity which is driven by the input metrics, e.g. the QoS costs, and the provided set of rules.

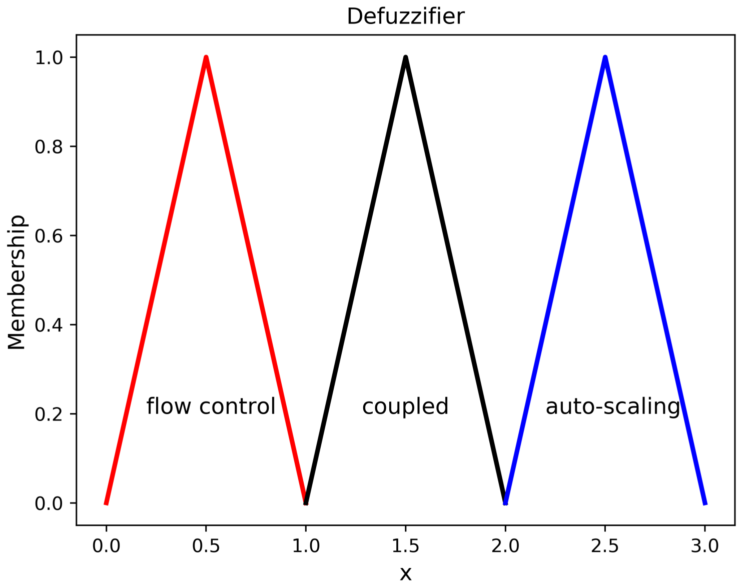


Figure 6.4.: Defuzzification logic.

A no operation state is not intended, since there is no clear transitional logic between the states. However, it could replace the coupled operation. In this setup, the fuzzy logic would decide to operate either auto-scaling, flow control or doing nothing.

### 6.5.3. Fuzzy Rules Set

The rule set is the engine of the fuzzy controller. In this section, we present a rule set which we obtain by reasoning.

The rule set should be provided by the service operator to be tailored to his needs. However, in the following, we present a simple mechanism, which decides to address an overload situation only with flow control or auto-scaling if the affected reconfiguration is too costly. With this, the burden of stabilizing the system is shared across both strategies in dependency to their resulting costs.

The first rule aims to activate the auto-scaler only, if the resources is good and the accuracy is bad, since this indicates, that the flow control mechanism has degraded the transmission rate to a wide degree in an overload situation, without having the auto-scaler provision resources:

```
RULE 1 : IF resources IS good and accuracy IS bad
THEN decision IS auto_scaling;
```

The second rules aims to activate flow control, if the accuracy is not degraded but the resources are bad, since this indicates, that the auto-scaler has already exhausted its options:

```
RULE 2 : IF accuracy IS good and resources IS bad
THEN decision IS flow_control;
```

The presented rule set is only an example, which demonstrates, how to share the burden of runtime management on two different strategy classes, such that the capabilities of each are promoted.

## 6.6. Discussion

In this section we have presented three coupling mechanisms, with different goals in mind. Whereas the concurrent coupling aims to combine runtime strategies in a transparent manner, the rule-based coupling utilizes a boolean algebra engine in order to support arbitrary thresholds based on binary evaluators. However, it requires a fine tuning of the thresholds in order to contribute to the QoS conformance. The fuzzy logic coupling allows to define linguistic rules which support blurry decisions. Therefore it is not limited to expressions, which evaluate as true, as the rule-based evaluator. Since we tailored the fuzzy rules coupling with QoS cost functions, we assume, that it is reusable for many applications, given, that the QoS cost functions are created in a similar manner.

# 7. SEIA – A Runtime Management Framework for Cloud Applications

In this chapter we introduce the **SE**lf-adaptive management framework for Cloud-**I**oT **A**pplications (SEIA)[1].

The objective is to provide a framework for researchers and service operators to create and deploy runtime strategies for Cloud-IoT applications across multiple cloud platforms including simulation environments. Therefore it provides abstract cloud and IoT concepts which can be instantiated for specific cloud technologies. Furthermore it offers a strategy manager to deploy and operate runtime strategies. It allows to be integrated into existing cloud environments. SEIA has been heavily used to conduct controlled experiments and case studies to empirically answer research questions in multiple publications [32], [33], [34].

The main contribution of SEIA is an architectural representation of Cloud-IoT applications with probes and effectors.

## 7.1. Overview

SEIA consists of three packages:

- **Managed System:** This package provides software architecture representations for Cloud-IoT applications. The abstractions are pragmatic and consider only cloud components, which are manageable by a runtime strategy in terms of observation or adaptation.
- **Managing System:** This package provides concepts to probe or reconfigure cloud nodes. Each representation of the managed system is associated with a predefined effector or probe. Furthermore, it contains core parts of self-adaptive systems, e.g. a continuous and discrete MAPE-K loop or a strategy controller for operating runtime strategies.
- **Technologies:** This package provides technology-specific implementations of the presented cloud concepts. In this context a technology is a (third-party) solution in the Cloud-IoT domain, e.g. RabbitMQ for message queues, or CloudFoundry as PaaS. Therefore, the package provides technology-specific implementations of node effectors or probes based on the API of the managed system.

[1] SEIA shares the name with a roman goddess, who protects the seeds once sown in the earth — instead of protecting the seeds once sown in the earth, it manages the Cloud-IoT Applications once deployed on the cloud.

## 7.2. Cloud Application Meta-Model

We adopt the representation of software architecture as a graph of interacting components presented in [31] and refine it to represent Cloud-IoT applications. Figure 7.1 illustrates the application meta-model. A cloud application consists of a set of cloud nodes with each node as a manageable component, e.g. a microservice. A cloud node is characterized by properties, which are either `observable` or `adaptable`. Probes and effectors of the framework bind to these properties. Cloud edges are the connectors and represent the interaction pathway between cloud nodes.

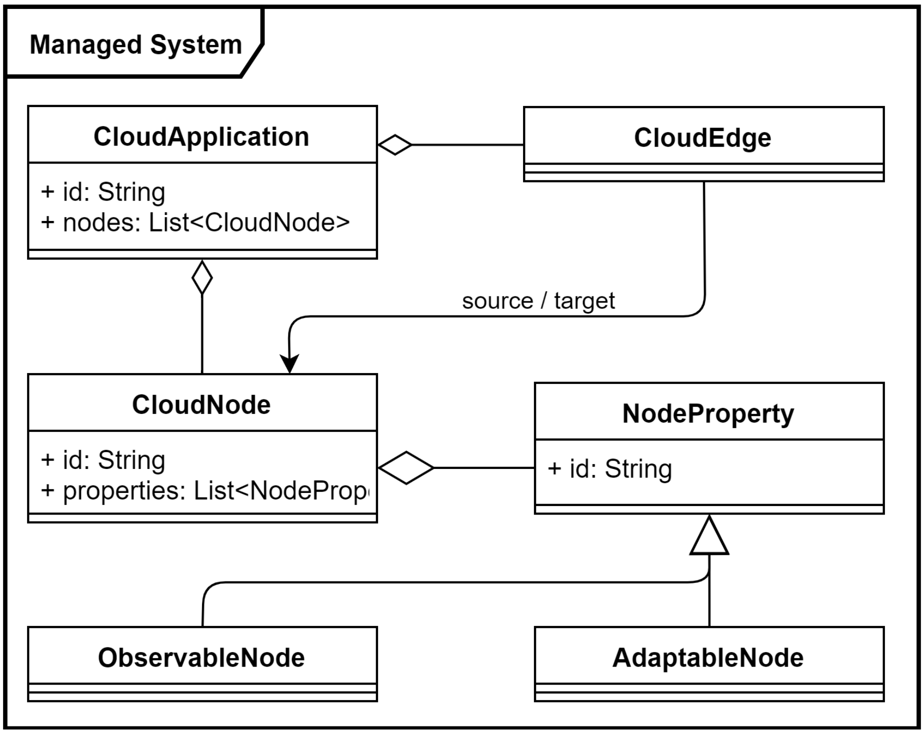


Figure 7.1.: Application meta model.

## 7.3. Cloud-IoT Concepts

The framework is tailored to support the application management in the Cloud-IoT domain. Conceptually, an IoT solution consists of three parts: the smart devices, which are sensing and effecting the environment, the IoT platform, which manages connected devices, and the cloud solution, which contains business logic and processes the data of smart devices [35]. `SEIA` provides a set of predefined classes for reoccurring components in this domain. These classes have to be extended to represent technology-specific entities and enriched with information needed to instantiate a concrete binding to the component. As illustrated in figure 7.2 we focus on the following components:

- **Smart Devices**: Sensor data from smart devices are the main driver of workload in this domain. In order to enable flow control approaches `SEIA` provides concepts for observable and adaptable smart devices. The node providing these functionalities is an IoT platform or an IoT data collection platform, since it manages connected smart devices. Therefore it is responsible to provide the capability to publish application-specific smart device statistics or to apply reconfiguration decisions.

- **Microservices**: Cloud solutions are typically realized using a set of interacting microservices. For this reason SEIA provides concepts for nodes, which provide system stats or are horizontal replicable.
- **Message Queues**: Message queues are modeled as a first class entity in SEIA since modern message brokers provide detailed statistics about the message queue state, which can be used by strategies to analyze the application state. Additionally, runtime approaches as presented in [29] are capable of scaling queues in a dynamic manner.

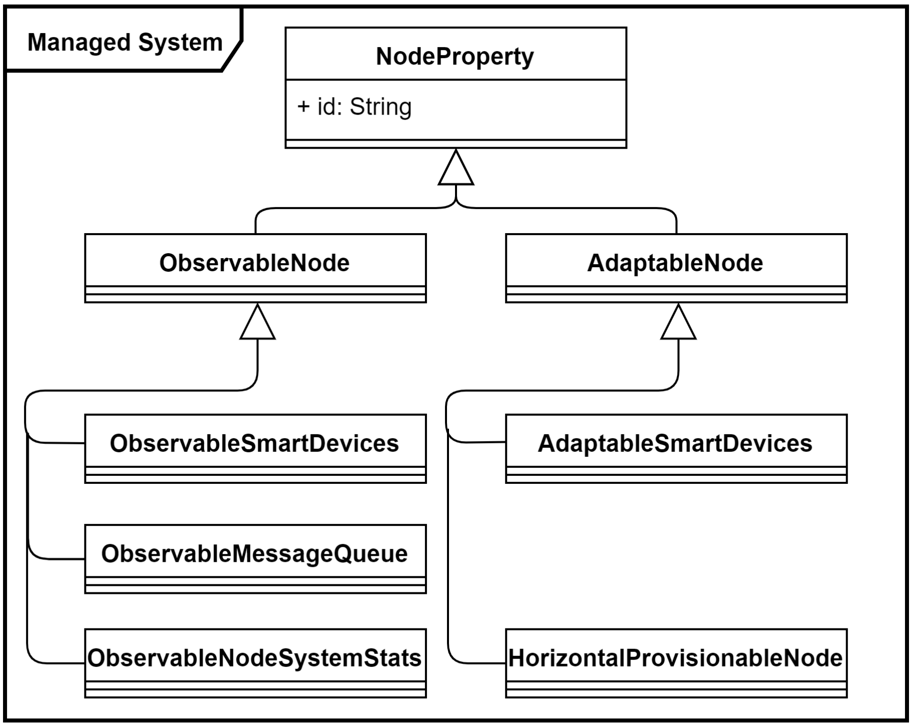


Figure 7.2.: Meta model of cloud concepts.

The presented concepts are not exhaustive and based on the requirements of the use cases managed by SEIA presented in chapter 8.3.

## 7.4. Probes and Effectors

Reusable probes and effectors are part of the managing system package and are associated with the presented node properties. SEIA strategies rely on observers or effectors on this abstraction level to manage the cloud application.

### 7.4.1. Probes

Figure 7.3 illustrates the model of probes. For each observable node one or more node observer are associated. The node observer references a specific observable node type and defines the data structure of measurements.

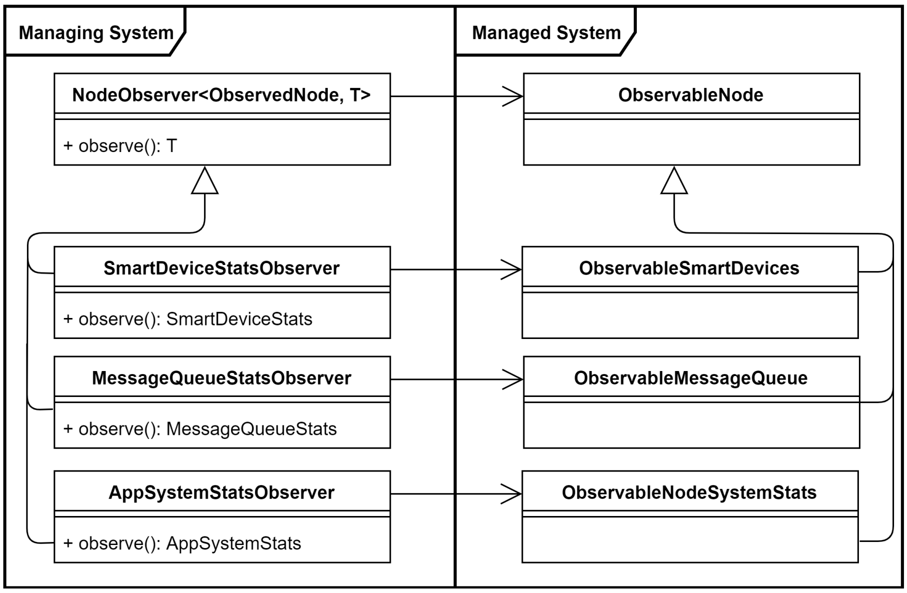

Managing System
Managed System
NodeObserver<ObservedNode, T>
+ observe(): T
ObservableNode
SmartDeviceStatsObserver
+ observe(): SmartDeviceStats
ObservableSmartDevices
MessageQueueStatsObserver
+ observe(): MessageQueueStats
ObservableMessageQueue
AppSystemStatsObserver
+ observe(): AppSystemStats
ObservableNodeSystemStats

Figure 7.3.: Class diagram of SEIA probes.

Whereas the entities contain the following fields:

- **SmartDeviceStats**: Number of active devices, average transmission rate.
- **MessageQueueStats**: Queue length, queueing delay (wait time), arrival and departure rate, growth, number of consumers.
- **AppSystemStats**: Number of active and planned replicas, aggreggated and per instance system metrics (CPU, memory, disk).

The managing system can be extended by further node observers which may rely on another data structure.

### 7.4.2. Effectors

Figure 7.4 illustrates the model of effectors. An effector is associated to an adaptable node property and provides a method to reconfigure it. SEIA provides an adaptation engine for smart devices to reconfigure the transmission rate and a scaler of horizontal provisionable nodes, which can be used to create auto-scalers.

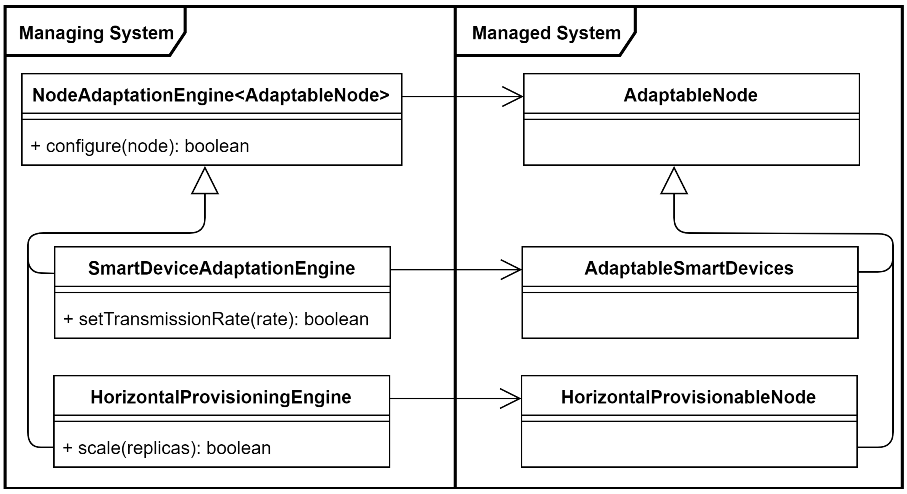

Managing System
Managed System
NodeAdaptationEngine<AdaptableNode>
+ configure(node): boolean
AdaptableNode
SmartDeviceAdaptationEngine
+ setTransmissionRate(rate): boolean
AdaptableSmartDevices
HorizontalProvisioningEngine
+ scale(replicas): boolean
HorizontalProvisionableNode

Figure 7.4.: Class diagram of SEIA effectors.

## 7.5. Monitoring Concept

SEIA provides a lightweight monitoring concept, which consists of a monitoring loop and $N$ monitoring tasks, whereas $N$ is the number of observed components. Figure 7.5 illustrates the monitoring model.

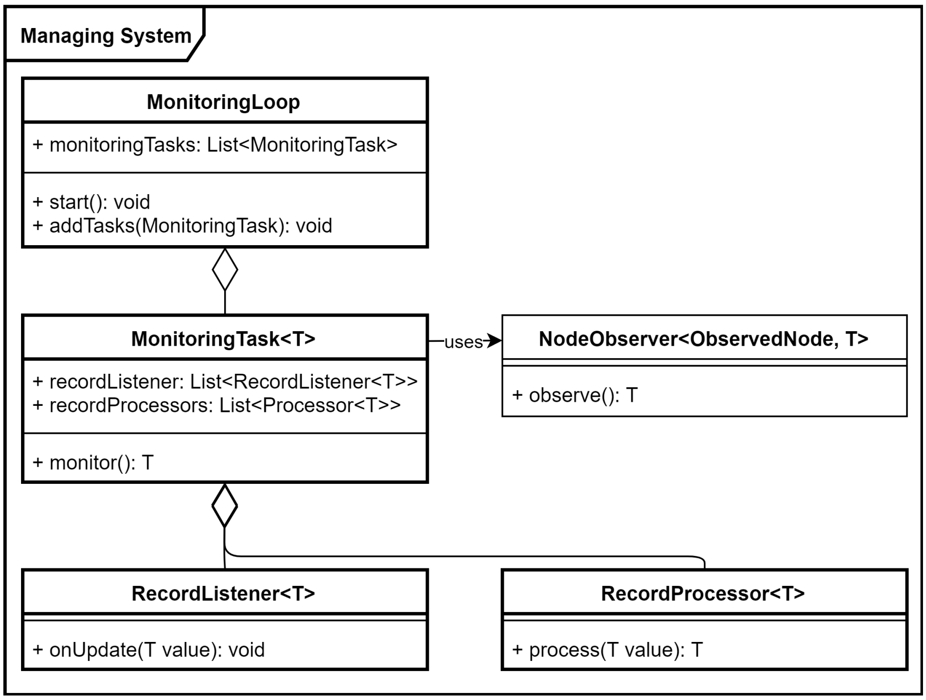

Managing System
MonitoringLoop
+ monitoringTasks: List<MonitoringTask>
+ start(): void
+ addTasks(MonitoringTask): void
MonitoringTask<T>
+ recordListener: List<RecordListener<T>>
+ recordProcessors: List<Processor<T>>
+ monitor(): T
uses
NodeObserver<ObservedNode, T>
+ observe(): T
RecordListener<T>
+ onUpdate(T value): void
RecordProcessor<T>
+ process(T value): T

Figure 7.5.: Class diagram of SEIA monitoring.

Measurements from observable nodes are usually obtained using external calls. External calls can be expensive in terms of response time or contract based expenses. Therefore, SEIA introduces monitoring tasks, which embed a node observer in order to retrieve current measurements once within fixed intervals. Runtime strategies should attach as listeners to the monitoring tasks to receive measurements. Monitoring tasks provide the possibility to attach listener and processors to it. A processor is a component, which is able to manipulate raw measurement data, e.g. to smooth or aggregate data. Attached listeners are called sequentially and receive the processed measurements.

Monitoring tasks are appended to a monitoring loop. The monitoring loop calls the monitoring task in a specific interval. The monitoring tasks are called asynchronously and the monitoring loop guarantees, that each task is called at max once during the specified monitoring interval. If the execution of a monitoring task exceeds the monitoring interval, it transits to the subsequent interval without any further delay.

## 7.6. Strategy Concept

Figure 7.6 illustrates the strategy model of SEIA, which provides a loose framework for strategies without imposing many design choices on it. A strategy is a reusable, technology-agnostic component, which provides life cycle methods for its creation and graceful deconstruction. Furthermore, it provides an update method, which is called by a strategy controller in certain intervals. It can be used as a heartbeat for the strategy, such that it triggers an analysis of the applications' state and the planning and execution of adaptation steps. However, for computational expensive strategies, a strategy should manage its own threads. The model of strategies does not enforce the MAPE-K reference architecture on its implementation, thus reducing the number of necessary classes per strategy, speeding up the prototypical development time on the expense of a conceptual separation of concerns. However, a strategy can be extended to follow the MAPE-K paradigm. In order to retrieve measurements, a strategy should register itself as a listener to the monitoring task which is observing the component of interest. A strategy can use probes defined in the previous section to reconfigure the system.

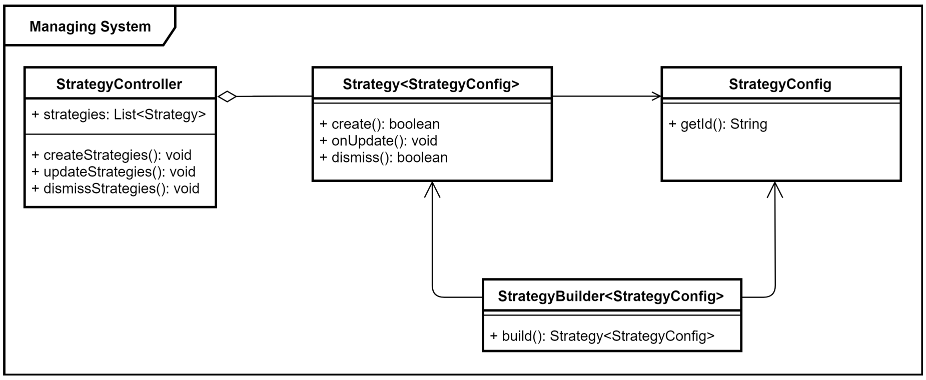

Managing System
StrategyController
+ strategies: List<Strategy>
+ createStrategies(): void
+ updateStrategies(): void
+ dismissStrategies(): void
Strategy<StrategyConfig>
+ create(): boolean
+ onUpdate(): void
+ dismiss(): boolean
StrategyConfig
+ getId(): String
StrategyBuilder<StrategyConfig>
+ build(): Strategy<StrategyConfig>

Figure 7.6.: Strategy Model of SEIA.

In order to encapsulate the strategy creation SEIA introduces strategy builders. A strategy builder encapsulates the construction knowledge of a strategy and is responsible to map relevant probes and effectors to the strategy. However, it is optional to instantiate a builder.

## 7.7. Binding Factory

SEIA introduces an binding factory in order to instantiate effectors and probes based on the underlying technologies of the cloud application. Figure 7.7 illustrates the factory with an example of instantiating the bindings of a cloud application deployed on a Kubernetes cluster.

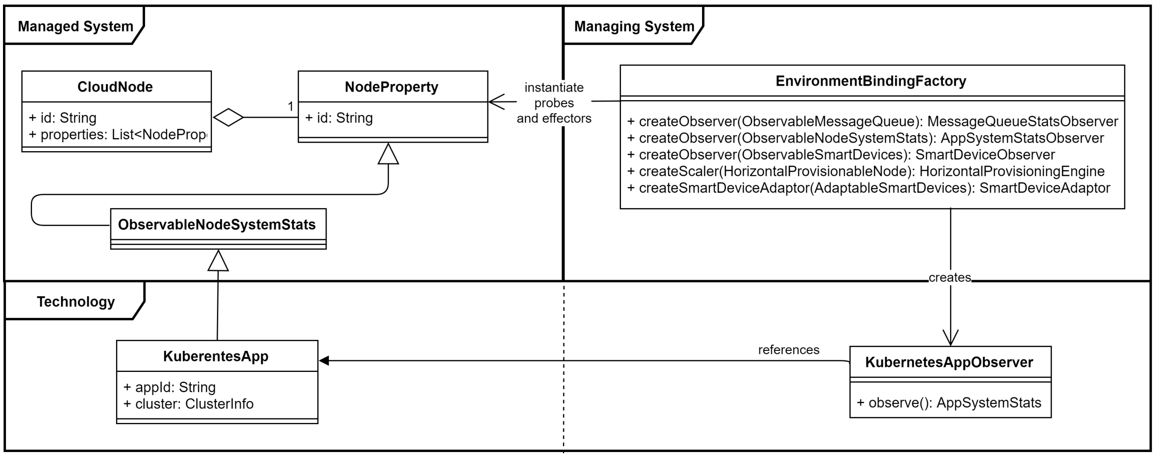

Managed System
Managing System
CloudNode
+ id: String
+ properties: List<NodeProp
1
NodeProperty
+ id: String
instantiate
probes
and effectors
EnvironmentBindingFactory
+ createObserver(ObservableMessageQueue): MessageQueueStatsObserver
+ createObserver(ObservableNodeSystemStats): AppSystemStatsObserver
+ createObserver(ObservableSmartDevices): SmartDeviceObserver
+ createScaler(HorizontalProvisionableNode): HorizontalProvisioningEngine
+ createSmartDeviceAdaptor(AdaptableSmartDevices): SmartDeviceAdaptor
ObservableNodeSystemStats
creates
Technology
KuberentesApp
+ appId: String
+ cluster: ClusterInfo
references
KubernetesAppObserver
+ observe(): AppSystemStats

Figure 7.7.: Environment Factory Model of SEIA.

The environment binding factory is an abstract entity, which maps the concepts used by SEIA to the managed technologies of the cloud application. A researcher or service operator has to provide both, a concrete environment binding factory and a technology-specific application model. This allows to support heterogeneous cloud platforms and technologies but induces effort since the environment binding factory is only reusable for different cloud applications deployed on the same environment.

## 7.8. Mapping to the MAPE-K Framework

The MAPE-K loop is a reference architecture of self-adaptive systems. Whereas SEIA provides a lightweight monitoring infrastructure, it is not an implementation of MAPE-K. Figure 7.8 illustrates the relation between both.

Conceptually, SEIA provides the means for monitoring cloud components and executing adaptations. A strategy is not obligated to realize a MAPE-K loop in terms of separated phases. However, it can be extended to do so. By sharing the monitoring and execution phase with the MAPE-K loop it can be easily integrated into a reference implementation. On the other side, it allows to speed up the prototypical development time, by providing a strategy structure, which can unify multiple phases without the need to split it up across multiple classes.

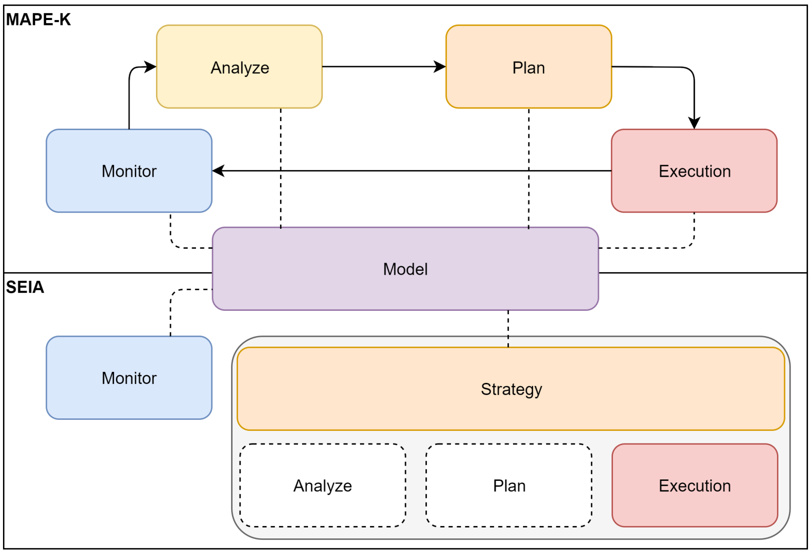


Figure 7.8.: SEIA and the MAPE-K loop.

## 7.9. Discussion

This section has presented SEIA, which is a framework for creating, deploying and operating runtime strategies on existing cloud applications. It is tailored to the Cloud-IoT domain and provides concepts for modeling, monitoring and adapting such systems. The contributions of SEIA are threefold: first, it allows to represent Cloud-IoT applications with a pragmatic view, which is based on the capabilities needed by a runtime management strategy. Second, it introduce concepts for probes and defines entities for reoccurring cloud components. Third, it provides concepts for effectors and define methods. It is lightweight in its nature, by providing loosely coupled components, which can be connected by the service operator, without enforcing many design choices.

Since the concepts are abstract, they can be instantiated for different cloud technologies. SEIA provides an built-in support for CloudFoundry, RabbitMQ and Kubernetes. It has been successfully applied to conduct research studies, ranging from auto-scaling, flow control to coupled strategies across multiple environments.

# 8. Validation

This chapter presents a validation of the contributions. We conduct a set of case studies to evaluate the central contributions of this thesis:

**C1:** Evaluation of the impact of capacity and resource demand variations on scaling decisions.

**C2:** Design of QoS cost functions for sensing cloud applications and a QoS optimization framework.

**C3:** Development of flow control approaches to overload protect sensing cloud applications.

**C4:** Design of rule-based coupling approaches to combine flow control with auto-scaling.

**C5:** Design of a sensing cloud application model for self-adaptive systems.

We rely on the Goal Question Metric (GQM) approach proposed by Basili et al. [4] to validate the contributions. It aims to orientate the validation alongside quantifiable and measurable metrics. The contributions are mapped onto the research questions presented in section 1.4.

The remainder of this chapters groups the case studies by the objective they intend to validate.

## 8.1. Validation Goals and Overview

We define for each contribution a validation goal based on the GQM approach. The GQM approach aims to hierarchically structure goals, questions and metrics, requiring to formulate each goal so it can be answered by collecting measurement data. Each goal clearly states the purpose of the validation, the issue to be measured, the measured object and the viewpoint from which the measurements is conducted.

### 8.1.1. GQM Plan

In this section we presents the validation goals. For each goal we state the corresponding research questions.

#### 8.1.1.1. Evaluation of Performance Metrics for Scaling Decisions

**Research Question 1.** *How robust are infrastructure metrics for scaling microservice to capacity and resource demand variations?*

**Goal 1 – Evaluation of Performance Metrics.** Evaluate the influence of capacity and resource demand variations on the quality of scaling microservices.

**Addressed RQs:** 1.

**Question 1.1.** How are the infrastructure metrics affected by capacity or resource demand variations?

**Metric 1.1.1.** Pearsons correlation coefficient between the infrastructure metrics and capacity and resource demand variations.

**Addressed RQs:** 1.

**Question 1.2.** How accurate is the analytical CPU utilization and message queue model?

**Metric 1.2.1.** Prediction accuracy as percentage difference between predicted and measured metric values for an observation period.

**Addressed RQs:** 1.

**Question 1.3.** How do capacity or resource demand variations affect the elasticity of threshold-based auto-scaling systems?

**Metric 1.3.1.** Influence on the elastic deviation to quantify the quality of scaling decisions.

**Metric 1.3.2.** Influence on QoS violations to quantify the quality of scaling decisions.

**Addressed RQs:** 1.

**Question 1.4.** How accurate is the simulation model?

**Metric 1.4.1.** Prediction accuracy of the elastic deviation as percentage difference between predicted and measured metric values for an observation period.

**Metric 1.4.2.** Prediction accuracy of QoS violations as percentage difference between predicted and measured metric values for an observation period.

**Addressed RQs:** 1.

#### 8.1.1.2. Congestion Avoidance Characteristic of time-driven Flow Control of Smart Devices

**Research Question 3.** *What are suitable flow control approaches to improve the QoS conformance of sensing cloud applications?*

**Goal 2 – Congestion Avoidance Characteristics of Flow Control.** Investigate the congestion avoidance characteristic of time-driven flow control approaches for sensing cloud applications in different load and capacity scenarios. The congestion avoidance characteristic hereby denotes the relation of capacity utilization and congestion-induced delays.

**Addressed RQs:** 2.

**Question 2.1.** How efficient is each strategy in utilizing the capacity and avoiding congestions in fixed capacity and connectivity scenarios?

**Metric 2.1.1.** Pareto Curve of average capacity utilization and average queueing delay of an observation period.

**Addressed RQs:** 2.

**Question 2.2.** How efficient is each strategy in utilizing the capacity and avoiding congestions in varying capacity scenarios?

**Metric 2.2.1.** Pareto Curve of average capacity utilization and average queueing delay of an observation period.

**Addressed RQs:** 2.

**Question 2.3.** How efficient is each strategy in utilizing the capacity and avoiding congestions in varying connectivity scenarios?

**Metric 2.3.1.** Pareto Curve of average capacity utilization and average queueing delay of an observation period.

**Addressed RQs:** 2.

**Question 2.4.** How accurate is the simulation model?

**Metric 2.4.1.** Prediction accuracy of the service utilization as percentage difference between predicted and measured metric values for an observation period.

**Metric 2.4.2.** Prediction accuracy of the queueing delay as percentage difference between predicted and measured metric values for an observation period.

**Addressed RQs:** 2.

#### 8.1.1.3. QoS Characteristics of isolated and coupled time-driven Flow Control Approaches in Overload Scenarios

**Research Question 3.** *What are suitable flow control approaches to improve the QoS conformance of sensing cloud applications?*

**Research Question 4.** *What are suitable coupling mechanisms to improve the QoS conformance of sensing cloud applications by combining flow control with auto-scaling?*

**Goal 3 – QoS Conformance of Flow Control in Overload Scenarios.** Evaluate the QoS characteristics of isolated time-driven flow control approaches in overload scenarios. The QoS characteristics are hereby defined as the mix of QoS costs in overload scenarios.

**Addressed RQs:** 3.

**Question 3.1** How are the QoS costs of isolated time-driven flow control approaches characterized in overload scenarios?

**Metric 3.1.1.** Data accuracy, timeliness and resource QoS costs for an observation period.

**Metric 3.1.2.** Cumulative QoS costs for an observation period.

**Addressed RQs:** 3.

**Question 3.2** How do the QoS characteristics differ from resource provisioning approaches?

**Metric 3.2.1.** Data accuracy, timeliness and resource QoS costs for an observation period.

**Metric 3.2.2.** Cumulative QoS costs for an observation period.

**Addressed RQs:** 3.

**Goal 4 – QoS Conformance of Coupled Flow Control in Overload Scenarios.** Evaluate the QoS characteristics of coupled time-driven flow control and resource provisioning approaches in overload scenarios.

**Addressed RQs:** 4.

**Question 4.** How are the QoS costs of coupled time-driven flow control approaches characterized in overload scenarios?

**Metric 4.1.** Data accuracy, timeliness and resource QoS costs for an observation period.

**Metric 4.2.** Cumulative QoS costs for an observation period.

**Addressed RQs:** 4.

**Goal 5 – Validation of Simulation Results on the Bosch IoT Cloud.** Evaluate the accuracy of the simulation model.

**Addressed RQs:** 3.

**Question 5.** How accurate is the simulation model?

**Metric 5.1.** Prediction accuracy of data accuracy, timeliness and resource QoS costs as percentage difference between predicted and measured metric values for an observation period.

**Metric 5.2.** Prediction accuracy of cumulative QoS costs as percentage difference between predicted and measured metric values for an observation period.

**Addressed RQs:** 3.

#### 8.1.1.4. QoS Contribution of isolated and coupled time-driven Flow Control Approaches in different Application Scenarios

**Research Question 3.** *What are suitable flow control approaches to improve the QoS conformance of sensing cloud applications?*

**Research Question 4.** *What are suitable coupling mechanisms to improve the QoS conformance of sensing cloud applications by combining flow control with auto-scaling?*

**Goal 6 – QoS Conformance of (coupled) Flow Control in different Application Scenarios.** Evaluate the QoS contribution of isolated and coupled time-driven flow control approaches in overload scenarios for different sensing cloud application scenarios. The cloud application scenarios are hereby defined as cloud applications demanding a high QoS conformance to specific dimensions, e.g. timeliness or accuracy. The QoS contribution refers to the achieved cumulative QoS costs across the overload scenarios in each application scenario.

**Addressed RQs:** 3, 4.

**Question 6.1.** How do the approaches perform in terms of QoS conformance in overload scenarios in a mixed application scenario?

**Metric 6.1.1.** Data accuracy, timeliness, resource and cumulative QoS costs for an observation period.

**Metric 6.1.2.** Percentage difference of cumulative QoS costs for an observation period.

**Addressed RQs:** 3, 4.

**Question 6.2.** How do the approaches perform in terms of QoS conformance in overload scenarios in a time-driven application scenario?

**Metric 6.2.1.** Data accuracy, timeliness, resource and cumulative QoS costs for an observation period.

**Metric 6.2.2.** Percentage difference of cumulative QoS costs for an observation period.

**Addressed RQs:** 3, 4.

**Question 6.3.** How do the approaches perform in terms of QoS conformance in overload scenarios in an accuracy-driven application scenario?

**Metric 6.3.1.** Data accuracy, timeliness, resource and cumulative QoS costs for an observation period.

**Metric 6.3.2.** Percentage difference of cumulative QoS costs for an observation period.

**Addressed RQs:** 3, 4.

**Question 6.4.** How do the approaches perform in terms of QoS conformance in overload scenarios in a cost-driven application scenario?

**Metric 6.4.1.** Data accuracy, timeliness, resource and cumulative QoS costs for an observation period.

**Metric 6.4.2.** Percentage difference of cumulative QoS costs for an observation period.

**Addressed RQs:** 3, 4.

**Question 6.5.** How accurate is the simulation model?

**Metric 6.5.1.** Prediction accuracy of data accuracy, timeliness and resource QoS costs as percentage difference between predicted and measured metric values for an observation period.

**Metric 6.5.2.** Prediction accuracy of cumulative QoS costs as percentage difference between predicted and measured metric values for an observation period.

**Addressed RQs:** 3, 4.

#### 8.1.1.5. QoS Contribution on the Example of an Industry-relevant Sensing Cloud Application

**Research Question 3.** *What are suitable flow control approaches to improve the QoS conformance of sensing cloud applications?*

**Research Question 4.** *What are suitable coupling mechanisms to improve the QoS conformance of sensing cloud applications by combining flow control with auto-scaling?*

**Goal 7 – QoS Contributions of (coupled) Flow Control on the Example of a Sensing Cloud Application.** Investigate the QoS contribution of isolated and coupled time-driven flow control approaches on the example of an industry-relevant sensing cloud application.

**Addressed RQs:** 3, 4.

**Question 7.** How do the approaches contribute to the QoS conformance in the *predictive maintenance* use case of Connected Heating?

**Metric 7.1.1.** Percentage difference of data accuracy, timeliness and resource QoS costs for an observation period.

**Metric 7.1.2.** Percentage difference of cumulative QoS costs for an observation period.

**Addressed RQs:** 3, 4.

#### 8.1.1.6. Characteristics of the TCP-inspired Flow Control

**Research Question 3.** *What are suitable flow control approaches to improve the QoS conformance of sensing cloud applications?*

**Goal 8 – Congestion Avoidance Characteristics of TCP-inspired Flow Control.** Investigate the characteristics of the AIMD-based TCP-inspired flow control approach in a centralized and distributed setup in terms of congestion avoidance and fairness.

**Addressed RQs:** 3.

**Question 8.1.** How is the fairness of transmission rate adaptations across each smart device affected by a varying load in distributed setup?

**Metric 8.1.1.** Jain's Fairness Measure for an observation period.

**Addressed RQs:** 3.

**Question 8.2.** How does a distributed setup affect the adaptation quality?

**Metric 8.2.1.** Percentage difference of the average service utilization for an observation period.

**Metric 8.2.2.** Percentage difference of the average queueing delay for an observation period.

**Addressed RQs:** 3.

### 8.1.2. Case Study Systems

This section presents the case study systems we use to validate the contributions of the thesis. We use the following systems:

- **ShapeShifter** – A single-service cloud application which is configured to represent services with different characteristics.
- **Connected Heating** – An innovative IoT solution, which connects heating units via an intelligent gateway with a cloud solution.

The case study systems are deployed on heterogeneous cloud infrastructures. The validation for the ShapeShifter case study is supported by a calibrated simulation model and aims for a high external validity.

### 8.1.3. Validation Coverage

This section presents the coverage of validation goals and the corresponding case studies. Overall, we validate the goals with two case study systems. All goals — except for goal 7 — are evaluated with the ShapeShifter case study, since it allows to represent services with different characteristics. We conclude the validation findings with goal 7 and rely on the Connected Heating case study system, which is based on a productively used smart home application of the Robert Bosch GmbH.

**Goal 1 – Evaluation of Performance Metrics.** The first goal aims at understanding the impact of capacity and resource demand variations on scaling decisions. We rely on the ShapeShifter case study since it is highly configureable, allowing us to adjust the computation and wait time spent for each message in order to systematically create these variations. This case study has no smart devices, instead we induce the workload by producing messages with a load driver. We measure the CPU utilization and the queue departure rate in order to compare it to an analytical model (Question 1.1). Then, we investigate the quality of scaling decisions in varying capacity and connectivity scenarios for threshold-based rules auto-scaling systems (Question 1.2).

**Goal 2 – Congestion Avoidance Characteristics of Flow Control.** This goal aims to investigate the congestion avoidance characteristics of a set of flow control approaches in the context of sensing cloud applications. On the example of the ShapeShifter case study, we analyze varying load and capacity scenarios, in which the approaches adapt the transmission rate in accordance to the capacity of the cloud service. Since applications differ in timeliness demands, we create a pareto curve consisting of the achieved utilization and the congestion-induced delay. This allows a comparison across the approach candidates.

**Goal 3 – QoS Conformance of Flow Control in Overload Scenarios.** The third goal aims to evaluate the influence of flow control approaches on the QoS conformance in overload scenarios. We introduce QoS cost sets for qualities of sensing cloud applications to the ShapeShifter case study and normalize the dimensions to each other. We investigate the influence of approaches on the QoS characteristics of the cloud application for varying congestion severities. We compare the approaches to each other via the cumulative QoS costs (Question 3.1). Furthermore, we compare them to state of the art resource provisioning approaches (Question 3.2).

**Goal 4 – QoS Conformance of Coupled Flow Control in Overload Scenarios.** The fourth goal extends goal 3 by investigating coupled approaches consisting of a combination of flow control and resource provisioning. Again, they are compared in terms of the cumulative QoS costs and the QoS cost characteristics in varying overload scenarios (Question 4).

**Goal 5 – Validation of Simulation Results on the Bosch IoT Cloud.** Goal 5 aims to validate the findings of goal 3 and 4 by comparing the predicted QoS costs to measured QoS costs on a Bosch cloud platform. It states the prediction accuracy for the cumulative QoS costs and the predicted costs of each dimension as a percentage error.

**Goal 6 – QoS Conformance of (coupled) Flow Control in different Application Scenarios.** The sixth goal investigates the capabilities of (coupled) flow control approaches to maintain the QoS in intensifying overload scenarios of time, accuracy and resource cost driven sensing cloud applications. In contrast to goal 3 and 4, we provide a fixed set of QoS cost function which is weighted to represent a specific application scenario. We create a range of intensifying overload scenarios and conduct the experiments on the ShapeShifter case study system. We compare them to each other and to resource provisioning approaches in terms of the cumulative QoS costs.

**Goal 7 – QoS Contributions of (coupled) Flow Control on the Example of a Sensing Cloud Application.** This goal investigate the contributions of the presented approaches on the QoS conformance on the example of a smart heating unit system. The system is based on a Bosch connected heating solution and deployed on a Kubernetes cluster. In contrast to the previous investigations, we capture the accuracy sensed by the smart devices. We compare the results of isolated and coupled approaches with two different cost function sets, of which one considers the perceived accuracy. The results are compared in terms of the cumulative QoS costs.

**Goal 8 – Congestion Avoidance Characteristics of TCP-inspired Flow Control.** The goal is to investigate the characteristics of AIMD-based TCP-inspired flow control approaches in a distributed and centralized setup. We compute the achieved fairness in a distributed setup (Question 8.1) and compare the adaptation quality of a distributed with a centralized setup (Question 8.2). The adaptation quality captures the average service utilization and the congestion-induced delay.

## 8.2. Experimental Setup

In this section we describe the experimental setup, which is illustrated in figure 8.1. The experiments are conducted via the `SEIA` framework which has been introduced in chapter 7. Since many of the experiments demand optimized candidates of the approaches to investigate their characteristics, we instantiate the optimization framework presented in chapter 4.

### 8.2.1. Overview

All experiments are conducted by using the `SEIA` framework, which has been extended by a case study controller.

**Experiment Controller** The experimental controller is the core part of each experiment. It configures the runtime strategies, the connectivity of the devices during the experiment and observes the system under test. The measurements are collected and analyzed in order to answer a given objective.

**System under Test** The system under test are sensing cloud applications, which provide monitoring and reconfiguration interfaces for their cloud components. The managed cloud components are microservices, message queues and smart devices. The runtime environment of the sensing cloud application is either a simulation model or a real cloud infrastructure. Since most of the validation goals demand highly optimized candidates we rely in most instances on a simulation model. The findings are then validated by conducting experiments on the IoT application deployed on the cloud infrastructure.

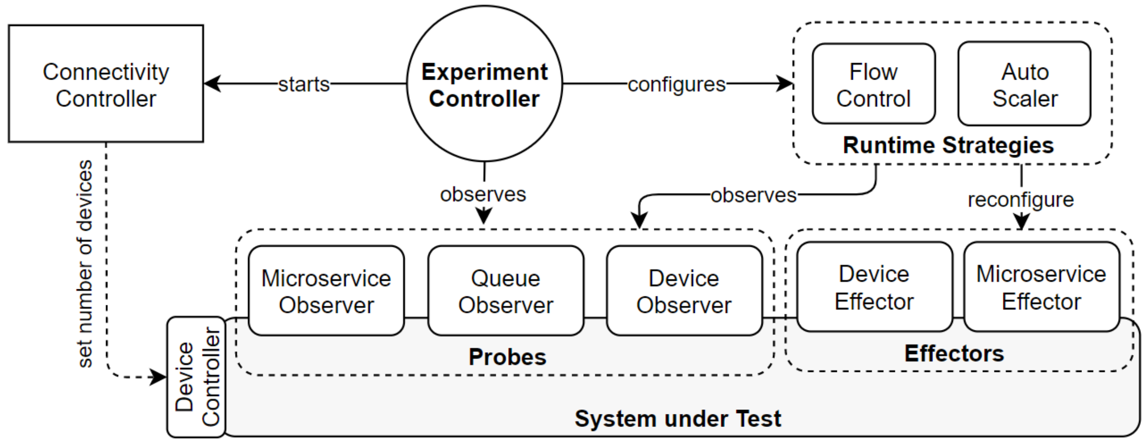


Figure 8.1.: Experimental Setup. The system under test is a case study system.

**Connectivity Controller** Connectivity patterns describe the changing number of connected devices over time. The workload on the system results from the number of connected devices and the configured transmission rate. Therefore the number of devices is given by the experimental setup whereas the transmission rate can be reconfigured during the experiment resulting in a dynamic workload. The connectivity controller uses a binding to the infrastructure of the system under test to create or destroy synthetic smart devices.

**Runtime Strategies** The runtime strategies monitor and reconfigure the system based on their internal adaptation engine, e.g. a flow control strategy or an auto-scaler.

### 8.2.2. Optimization Framework

In the following we instantiate the optimization framework introduced in chapter 4.6. For each validation goal, we provide a custom set of QoS cost functions, which are introduced and discussed in the experimental setup. In the following we introduce the simulation model and the optimization method used across all validation goals, which require an optimization.

### 8.2.3. Simulation Model

As a simulation model we rely on the QT based model introduced in chapter 3.2. Whereas it is limited in not supporting multiple cloud services, it allows to simulate the dynamic reconfiguration of the provisioning and the transmission rate. In this model, we obtain the queueing delay measured by the message queue as the input for a timeliness cost function or perceived accuracy. We observe the transmission rate of each smart device as an input for an accuracy cost function based on the collection interval. We model the microservice of the cloud application with a service time distribution, in order to simulate processing time. Overall, the simulation model allows to measure the QoS metrics introduced in section 4.3. Therefore, the results can be used to quantify the QoS costs achieved by a specific configuration. It enables to simulate runtime management approaches by providing a probing and effecting infrastructure. Therefore, it provides the interfaces required for an integration into the optimization framework presented in section 4.6. For each validation goal, the results are validated against measurements on an implemented system.

#### 8.2.3.1. Optimization Method

We optimize candidates based on a differential evolution (DE) based search approach. DE has been presented in [69] and is a heuristic, multi-dimensional genetic optimization method. It optimizes a problem by maintaining a population of candidate solutions which are combined to existing ones and filtered in terms of the best fitness, i.e. lowest QoS costs. We select DE due to being able to be parallel computed, to optimize multiple objectives and its widespread usage in academia and industry. Furthermore we assume a nonlinear objective function with many variables based on the potentially high parameter space of runtime management approaches. The parameters of runtime candidates are observed as dimensions, which have to be optimized. As a fitness function, we rely on the cumulative QoS costs.

## 8.3. Case Study Systems

In the following, we present the case study systems used for validating the environments. They are deployed on heterogeneous cloud infrastructures and each experiment is managed by `SEIA`.

### 8.3.1. ShapeShifter

The `ShapeShifter` case study system aims for a high external validity by providing a highly configurable IoT solution consisting of smart devices, a messaging middleware and a cloud service.

#### 8.3.1.1. Architecture

Figure 8.2 illustrates the conceptual architecture of the system.

**Microservice.** The microservice consumes messages out of a message queue and processes each message in a FIFO manner with characteristics based on its configuration. The configuration of the microservice allows to set the computation and I/O wait time for processing a message. The sum of both impacts the capacity of a microservice instance since it determines the service time, whereas the mix of both determines its resource characteristics. We implement the microservice as a lightweight Python application which provides a REST API as reconfiguration interface. The configuration is shared across all microservice instances using a network file system (NFS). We do not rely on environment variables, since the cloud platform does not support a dynamic change of environment variables. Based on timestamps in messages the microservices also publish the current end-to-end time.

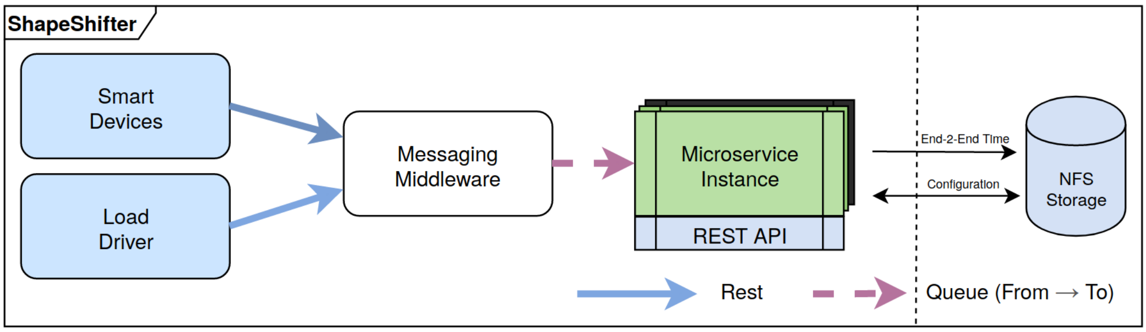


Figure 8.2.: Illustration of the ShapeShifter case study. The computation steps and the wait time on the microservice can be adjusted to variate the microservices' characteristics.

**Messaging Middleware (IoT platform).** The messaging middleware provides a REST API to receive messages from devices or a load driver. These messages are then enqueued in a message queue and processed by the microservice. The messaging middleware acts as a lightweight IoT platform, since it allows smart devices to connect and receives sensor data. It also offers an interface to adjust the transmission rate of each device. Furthermore, the communication to each smart device can be delayed by a given latency distribution, in order to attribute to effects in a productive environment.

**Smart Devices.** Due to cost reasons, the smart devices are synthetic entities, which are instantiated by a controller of the IoT platform. A smart device is an entity, which periodically sends sensor data with a creation timestamp in the payload. The timestamp allows to track the passed time between the creation and processing of a message.

**Load Driver.** The load driver creates messages and provides them to the messaging middleware. The load driver produces the messages with a time-varying intensity based on workload patterns and has no smart device semantic.

#### 8.3.1.2. Environment

The ShapeShifter case study system is deployed on the Bosch IoT Cloud, a PaaS based on Pivotal CloudFoundry. We extend `SEIA` with bindings to CloudFoundry and obtain application stats, e.g. CPU utilization, using the official CloudFoundry Java client[1]. Furthermore we implement a CloudFoundry horizontal provisioning engine in order to be able to perform scale in or out operations. We select Pivotal RabbitMQ as a message broker, which we deploy on a VM within the cloud infrastructure. We extend `SEIA` with a RabbitMQ binding based on the official management API. Furthermore we install a message timestamp plugin[2], in order to retrieve the queueing delay of a given queue. The smart device controller is also deployed as a microservice on the Bosch IoT Cloud.

[1] https://github.com/cloudfoundry/cf-java-client

[2] https://github.com/rabbitmq/rabbitmq-message-timestamp

#### 8.3.1.3. Simulation Model

We parameterize the simulation model presented in section 3.3 to represent the environment characteristics of this case study. In order to simulate the monitoring behavior of RabbitMQ and CloudFoundry, we conduct measurements on our cluster to parameterize the measurements providers $MP = (T_X, N)$ as a tuple of an update interval $T_X$ and a moving average of size $N$ as follows:

- Microservice on CloudFoundry: $MP_{Microservice} = (15\,\text{sec}, 15)$
- Message Queue on RabbitMQ: $MP_{MessageQueue} = (5\,\text{sec}, 10)$

Furthermore, we measure the provisioning time for a single ShapeShifter microservice on CloudFoundry and set the scaling delay $T_{Scaling}$ to 3 sec.

### 8.3.2. Connected Heating

In this section, we present Connected Heating, an innovative IoT solution of the Robert Bosch GmbH. By connecting heating units via an intelligent gateway with cloud solutions it supports a number of use cases. The case study is used to evaluate the impact of runtime management approaches presented in this thesis on the QoS conformance on the example of an industry relevant system.

#### 8.3.2.1. Architecture

The heating units are connected via a messaging middleware to the cloud solutions. Figure 8.3 illustrates the system.

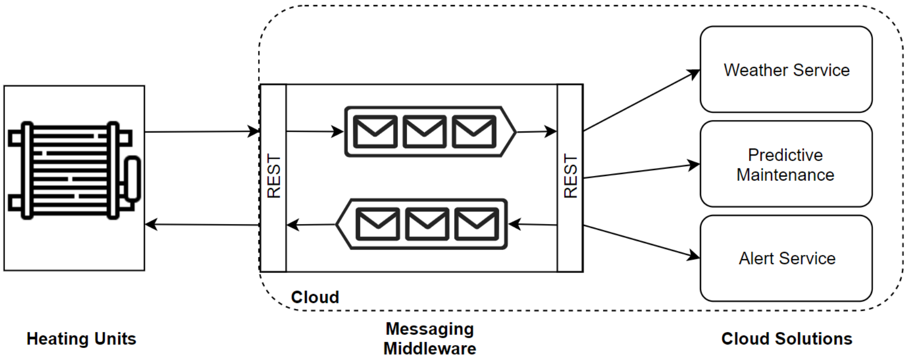


Figure 8.3.: Connected Heating case study system.

Connected heating units communicate via a REST API to the messaging middleware. The messaging middleware enqueues received messages and dispatch them to corresponding cloud solutions which may consist of multiple services. Each cloud solution retrieves messages via a dedicated message queue. Whereas Connected Heating supports a range of use cases we focus on the `Predictive Maintenance` use case. In this use case heating units periodically send pressure and temperature data which is analyzed by a predictive maintenance service.

#### 8.3.2.2. Implementation

Connected Heating is based on a productively used system which we re-implement. The cloud solutions consists of a scalable microservices with a configurable service time to simulate the business workflow, e.g. calls to the external database. Furthermore it consumes messages out of the RabbitMQ message queue. We extend the microservice and IoT platform implementations of the `ShapeShifter` case study. We rebuild the architecture of each system by deploying the configured microservices and creating the communication mechanisms. Furthermore, if applicable, we provide environmental models in order to enable sensing accuracy quantifications. Therefore the synthetic smart devices pass messages containing environmental data to the cloud application. All components are deployed on the Bosch IoT Cloud.

**Parameterization.** In cooperation with the Bosch business area we have conducted load tests on the Connected Heating system. These measurements contain message queue and microservice statistics which allows us to estimate the resource demand for each message using LibReDe [68]. Based on the estimated resource demand, we parameterize instances of the configurable microservice introduced in the previous case study system.

#### 8.3.2.3. Environment

The case study is deployed on a Kubernetes cluster hosted on the bwCloud[3], an IaaS for science and education. We extend `SEIA` with bindings to Kubernetes and obtain application stats, e.g. CPU utilization, using the Kubernetes CLI[4]. Furthermore we implement a Kubernetes deployment scaler in order to be able to perform scale in or out operations by replicating pods of a specific deployment. We select Pivotal RabbitMQ as a message broker and deploy it on the Kubernetes cluster.

#### 8.3.2.4. Predictive Maintenance Architecture

Figure 8.4 illustrates the cloud solution for the predictive maintenance use case.

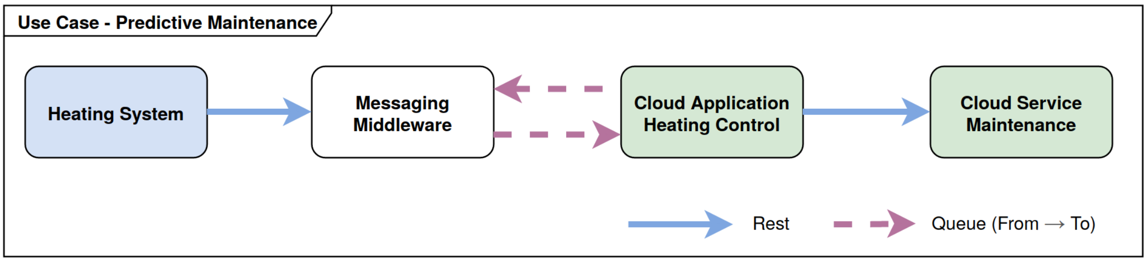

Use Case - Predictive Maintenance
Heating System
Messaging Middleware
Cloud Application Heating Control
Cloud Service Maintenance
Rest
Queue (From → To)

Figure 8.4.: Connected Heating case study system.

[3] https://www.bw-cloud.org/
[4] https://github.com/kubernetes/kubernetes/tree/master/pkg/kubectl

Heating units send periodically pressure and temperature sensor data to the heating control service. The heating control services stores the received data by providing it to a maintenance service.

The request happen regularly, at an interval of 15 sec per heating unit, inducing a constant load. The use case allows to investigate varying transmission patterns based on different policies.

## 8.4. Evaluation of Performance Metrics for Scaling Decisions

In this section, we evaluate the impact of capacity and resource demand variations on scaling decisions based on the ShapeShifter case study introduced in section 8.3. Furthermore, we validate the prediction accuracy of the analytical and simulation model presented in chapter 3. The validation targets Goal 1 of this thesis and enables a service operator to make an educated selection of the auto-scalers' performance metric for specific microservices.

### 8.4.1. Experimental Design

Figure 8.5 illustrates the experimental design. First, we calibrate the simulation model by conducting controlled experiments. Then we select the objective, e.g. evaluating the impact of capacity variations on the CPU utilization. We evaluate the performance metrics analytically by adjusting the resource demand and capacity. We use a simulation model to analyze the elasticity of auto-scalers based on these performance metrics and their robustness to capacity and resource demand changes. In order to search for optimal auto-scaler configurations, we instantiate the optimization framework introduced in section 8.2.2 with SEIA. We validate the predictions of the elasticity and the metrics by executing controlled experiments on the case study system deployed on the Bosch IoT Cloud.

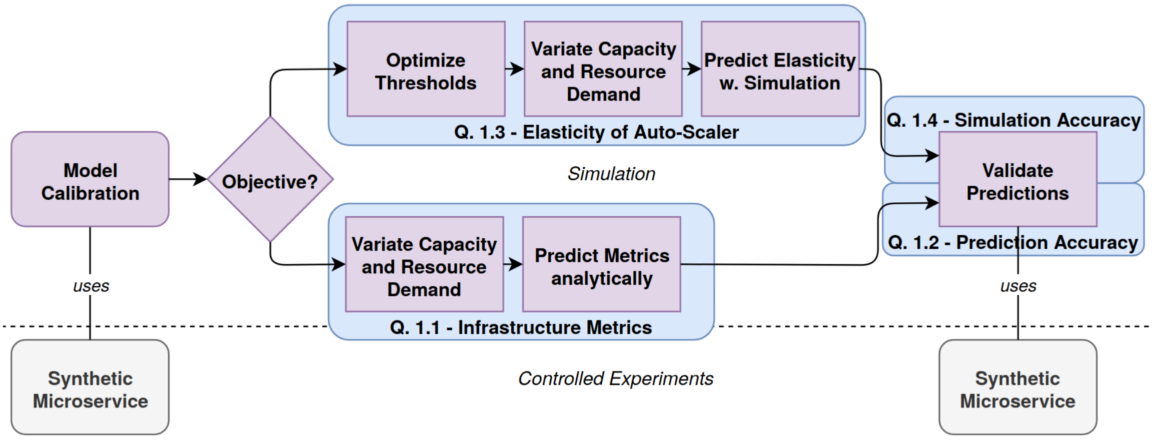

Optimize Thresholds
Variate Capacity and Resource Demand
Predict Elasticity w. Simulation
Q. 1.3 - Elasticity of Auto-Scaler
Q. 1.4 - Simulation Accuracy
Model Calibration
Objective?
Simulation
Validate Predictions
Q. 1.2 - Prediction Accuracy
Variate Capacity and Resource Demand
Predict Metrics analytically
Q. 1.1 - Infrastructure Metrics
uses
uses
Synthetic Microservice
Controlled Experiments
Synthetic Microservice

Figure 8.5.: Sequence Diagram of the experimental design.

**Addressing Question 1.1** We evaluate the impact of capacity and resource demand variations on infrastructure metrics by varying the resource demand of the microservice in a manner, which reflects variations in the overall service time and in the workload mix. In this setup, we do not utilize an auto-scaling system. Finally, we analyze the correlation between the infrastructure metrics to variations in the overall service time and the workload mix.

**Addressing Question 1.2** We compare the predictions of the calibrated queueing theory based model with measurements obtained by experiments conducted on the ShapeShifter case study. We quantify the correlation using the Pearson correlation coefficient [12].

**Addressing Question 1.3** In order to evaluate the impact on the auto-scalers elasticity we capture timing and accuracy aspects of scaling decisions and observe time-based QoS violations. We select an initial scenario for which we heuristically optimize each threshold. We use the optimized thresholds to evaluate the robustness of the auto-scalers in scenarios with varying service time and workload mix.

**Addressing Question 1.4** We compare the simulation predictions with measurements obtained by experiments conducted on the Bosch IoT Cloud infrastructure. We quantify the prediction accuracy for the performance metrics and the elasticity.

#### 8.4.1.1. Calibration

In this section, we briefly state the calibration approach to parameterize the analytical and simulation model.

**Calibrating the Analytical Model.** We set the resource demand of a message to $D_M = (D_{CPU})$ by disabling the I/O-share of processing a message. We configure the required computation steps $k$ to use it as baseline CPU demand $D_{CPU} = k$. The calibration module measures the round trip $T_{rtt}$ to the microservice. We conduct $i = 1000$ requests and measure the elapsed time $T_{ela}$ to retrieve the service time per request $ST_{Req} = \frac{T_{ela}}{i} - T_{rtt}$. Finally, we calculate the CPU capacity $C_{CPU} = \frac{ST_{Req}}{D_{CPU}}$. Using the capacity and a given CPU resource demand $D_{CPU}$ we predict the service time $ST_{CPU}$ with $ST_{CPU}(D_{CPU}) = C_{CPU} * D_{CPU}$.

**Calibrating the Simulation Model.** We calibrate the scaling delay $T_{Scaling}$ by conducting scaling operations and measure the time span between decision and impact. To calibrate the measurement providers we conduct a set of controlled experiments with varying workload intensity and patterns. By analyzing changes in the measurements, we identify the publish interval $T_X$. We model each measurement provider as a moving average of size $N$. We estimate $N$ by analyzing the measurements with expected values based on the controlled setup.

#### 8.4.1.2. Elasticity Quantification

To quantity the achieved elasticity of a metric-based auto-scaler we rely on elasticity metrics proposed in [37]. Based on a theoretical optimal behavior, in which the resource demand is matched by the supplied resources, we calculate the timeshare spent in an under- or overprovisioned state: $\tau_U$ and $\tau_O$. To capture accuracy aspects, we rely on the relative amount of under- and overprovisioned resources: $\theta_U$ and $\theta_O$. We aggregate the set of elasticity metrics by computing the elastic deviation $\sigma$. In contrast to the elastic speedup, which is also proposed in [37], the elastic deviation does not require a no-scaler baseline to quantify the elasticity. It expresses the resemblance to an optimal auto-scaling with a value of 0 as an ideal match.

#### 8.4.1.3. Optimize Thresholds

A threshold-based rules auto-scaling systems typically relies on an upper and lower threshold for scaling decisions. Therefore, the selection of the thresholds has a strong impact on the achieved elasticity.

We formulate the optimization problem as follows: Let the configuration of the auto-scaler consists of a performance metric $N_k = \{CPU, QLength, QDelay, QGrowth\}$ with a threshold pair $M = \{lower, upper\}$. Let $C_k : N_k x M \rightarrow \Re$ an optimal mapping such that the objective $F = \sigma$ is minimized. The computed elastic deviation $\sigma$ is used as fitness function in the optimization framework.

#### 8.4.1.4. Capacity and Resource Demand Variations

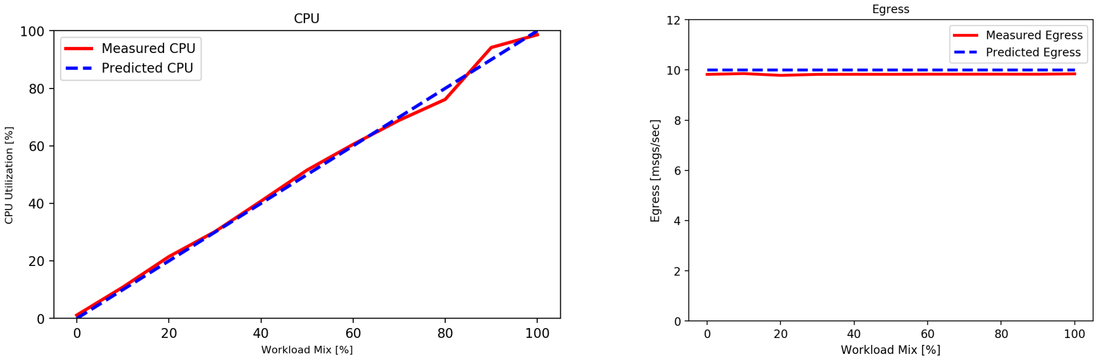
We model the impact of capacity or resource demand variations by changing the fraction of compute to wait operations — denoted as workload mix — or the overall service time for processing a message.

**Workload Mix.** Let a workload mix be in the range of 0 % to 100 % and expresses the share of compute to wait time. The workload mix $L_{mix}$ is determined by adjusting the resource demand $D_{CPU}$ and $D_{IO}$ such that $L_{mix} = \frac{D_{CPU} * C_{CPU}}{D_{CPU} * C_{CPU} + D_{IO} * C_{IO}}$

### 8.4.2. Q.1.1 – Impact of Resource Demand and Capacity Variations on Infrastructure Metrics

In this section we evaluate the impact of workload mix and service time variations on CPU and message queue metrics. We use the analytical model presented in chapter 3. We produce a workload burst of 600 messages with a size of 500 Bytes to the message queue. The experiment duration is 2 min.

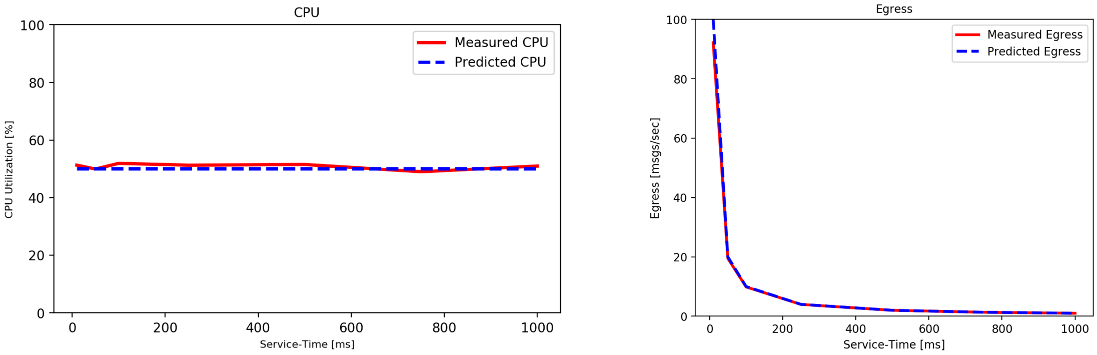
**Workload Mix Variations.** We evaluate the impact of changes in the workload mix on the infrastructure metrics, whereas the overall service time does not change. We adjust the workload mix $L_{mix}$ in a range of 0 % to 100 % by adjusting the relation of $D_{CPU}$ and $D_{IO}$. The overall service time is 100 ms per message in each scenario.



(a) CPU Utilization

(b) Output Rate

Figure 8.6.: Measured and predicted CPU utilization and queue output rate for a varying workload mix.



(a) CPU Utilization

(b) Output Rate

Figure 8.7.: Measured and predicted CPU utilization and queue output rate for a varying service time.

Figure 8.6 shows the predictions over the variations in the workload mix. The results show a strong correlation of the CPU utilization to the workload mix of 0.99. There is no correlation between queueing metrics to the workload mix.

**Service Time Variations.** We evaluate the impact of changes in the service time on the infrastructure metrics, whereas the workload mix stays constant at 50 %. We vary the overall service time per message between 5 ms and 1000 ms by proportionally increasing the resource demand $D_{CPU}$ and $D_{IO}$.

Figure 8.7 shows the predictions over the variations of the service time. The results show a correlation of the queue output rate of −0.59. The correlation is degraded since the output rate is the inverse of the service time, resulting in a non-linear relation. The CPU has no correlation to the service time.

**Conclusion.** We analyzed the impact of resource demand and capacity variations on the CPU and queueing metrics. We calibrated a queueing theory based model to predict the CPU utilization and the queue output rate of messages. The predictions show, that the CPU utilization correlates with the workload mix, whereas queueing metrics correlates with service time variations.

### 8.4.3. Q.1.2 – Infrastructure Metric Model Accuracy

In this section we evaluate the prediction accuracy of the analytical model by conducting controlled experiments on the Bosch IoT Cloud. Figure 8.6 shows the predictions for a varying workload mix. The prediction error of the CPU utilization is 10.72 % and for the queue departure rate −1.74 % respectively. Since our model does not include the fetching of messages out of the message queue, we expect that this is responsible for the slightly degraded departure rate and the CPU degradation due to an increased service time. Figure 8.7 shows the predictions for a varying service time. The prediction error of the CPU utilization is 1.62 % and for the queue departure rate 1.93 % respectively. The analytical model predicts the CPU utilization and the queue departure rate with a high accuracy.

### 8.4.4. Q.1.3 – Impact of Resource Demand and Capacity Variations on Scaling Decisions

This sections evaluates the impact of resource demand and capacity variations on scaling decisions of threshold-based auto-scalers. We capture the elasticity of the auto-scalers in two scenarios: first, by adjusting the service time, then by adjusting the workload mix. We evaluate the elasticity of the auto-scalers based on a simulation model. Therefore, we refine the experimental design.

#### 8.4.4.1. Experimental Design

We conduct two sets of experiments. First, we variate the workload mix by adjusting the CPU share for each message in a range of 0 % to 100 %. Furthermore, we variate the service time in a range of 50 ms to 1000 ms. We capture the elasticity measures to evaluate the auto-scalers in each scenario. Furthermore, we obtain SLO metrics, such as the percentage of response time violations with a threshold of 1000 ms. We derive the response time by measuring the queueing delay. Furthermore, we include the average number of instances. We configure the differential evolution with 80 iterations and a population size of 10. The optimization controller uses the simulation model to evaluate a given auto-scaler configuration. We compare the optimized auto-scalers with a baseline which is a no-scaler scenario with a statically provisioned microservice. Finally, we validate the simulation findings by conducting controlled experiments with the optimized threshold configuration across some points of the given scenarios. For each measurement we compare the elastic deviation and the SLO violations in order to confirm the predictions of the simulation.

**Application Scenario.** In the initial application scenario, we configure the service time per message as follows: $ST_M = ST_{CPU} + ST_{IO} = 500\,\text{ms}$ with $\frac{ST_{CPU}}{ST_{IO}} = 1$ such that the CPU share for each message is 50 %. The microservice is provisioned with 1 instance.

**Workload.** We use traces[5] from the 24.12.2007 of the english Wikipedia to obtain a real-world workload with time-varying behavior. The traces are on a per hour base, which we speed up by factor 60, such that each hour is a minute in the experiment. Furthermore we decrease the load intensity by a factor of 2500. The overall experiment duration is $T_{exp} = 24\,\text{min}$ with a total of 13800 produced messages.

**Simulation Model.** The simulation model uses the calibrated analytical model to calculate the service time and CPU utilization in each simulation step. Furthermore, we calibrate the simulation model to reflect the monitoring policies of the measurement provider.

**Elasticity Quantification.** The elastic deviation is used as fitness function for the optimization and is part of the elasticity metrics used to evaluate the auto-scalers. We adjust the elastic deviation to exclude instability, since there are no costs associated with unnecessary resource supply adaptations on our PaaS. Therefore we calculate the elastic deviation as follows: $\sigma = (\theta_U^4 + \theta_O^4 + \tau_U^4 + \tau_O^4)^{\frac{1}{4}}$.

#### 8.4.4.2. Results

In the following we evaluate the auto-scalers in three different scenarios. In the initial application scenario we discuss the results achieved by the optimized thresholds. Then, we vary the service time and the workload mix to evaluate the impact of each on the elasticity of the auto-scalers and the SLO violations.

**Initial Application Scenario.** Table 8.1 summarizes the elasticity and SLO metrics of the optimized thresholds for the initial application scenario. In terms of elasticity the queue growth excels the other metrics whereas the CPU utilization is the worst. In terms of the SLO violations the CPU utilization excels the other metrics with the queue growth as the worst. The CPU utilization provides a balanced mix between elasticity and SLO violations. The queue growth is able to adjust provisioned resources in a fine-granular manner by considering the current capacity to the current load. However, it is barely affected by the current queue length resulting in a high percentage of SLO violations. The queue length is a metric which requires enqueued message in order to make scaling decisions. This results in a high amount of SLO violations based on the high service time per message and the strict SLO threshold. However, it performs well in regard to achieved elasticity. The queueing delay offers a similar elasticity as the queue length, but has a worse response time. Since all metrics are optimized towards elasticity, they favor an overall low deviation to the optimal scaling behavior over a low response time.

[5] $http://www.wikibench.eu/?page_id = 60$

| | No Scaling | CPU | Queue Length | Queue Growth | Queue Delay |
|---|---|---|---|---|---|
| $accuracy_u$ | 65.75 | **4.32** | 7.39 | 4.55 | 6.75 |
| $accuracy_o$ | 0 | 10.29 | 6.16 | **1.31** | 6.77 |
| $timeshare_u$ | 83.26 | **18.89** | 24.89 | 22.29 | 26.88 |
| $timeshare_o$ | 0 | 30.9 | 21.18 | **4.38** | 23.47 |
| $\sigma(elasticdev.)$ | 96.15 | 31.93 | 26.59 | **22.32** | 30.25 |
| SLO violations [%] | 99.72 | **22.15** | 57.78 | 88.19 | 70.14 |
| Avg. #Instances [#] | 1 | 5.34 | 5.15 | **4.9** | 5.06 |
| Avg. resp. time [sec] | 531.44 | **1.46** | 6.33 | 25.25 | 9.16 |
| Med. resp. time [sec] | 517 | **0** | 3 | 26 | 9 |
| elastic speedup | 1 | 2.74 | 3.01 | **5.92** | 2.64 |

Table 8.1.: Elasticity and SLO metrics for auto-scalers using a specific performance metric with optimized thresholds.

**Service Time Variations.** The impact of service time variations on the elastic deviation for each metric-based scaler is shown in figure 8.8, whereas the effect on the SLO is shown in figure 8.9. All metrics seems to be affected in a similar manner in terms of the elastic deviation. With decreasing service time they perform better since less resources and therefore scaling decisions are required. With an increasing service time the graph shows a slight increase in the elastic deviation since more resources and therefore scaling decisions are required. The queue growth and the queueing delay exhibit a high variance. The queue length and CPU utilization a low variance. This indicates, that the queue growth and delay have a narrow space of high-performing configurations and therefore require a deeper search.

Both, the CPU utilization and the queue length, are able to maintain a low percentage of SLO violations. The queue growth exhibits an unsteady behavior which we assume to be based on the complex interaction of queue length, arrival rate and queue output rate. The queueing delay maintains a steady percentage of SLO violations. However, for an increasing service time the SLO violations are slightly decreasing which may be based on the faster growing queueing delay since the microservice processes messages slower with an unchanged arrival rate.

In conclusion, each metric has shown a high degree of robustness towards changes in the service time. The characteristics of the optimized thresholds are largely maintained during the variations, e.g. the elastic deviation or the percentage of SLO violations. The CPU utilization offers in this setup a good mix of elastic deviation and SLO conformance, followed by the queue length.

**Workload Mix Variations.** Figure 8.10 shows the effects of workload mix variations on the elastic deviation, whereas figure 8.11 shows the effect on the SLO. As concluded in the previous evaluation, the message queue metrics are not affected by the workload mix. This results in a constant elasticity across the workload mix variations. In contrast, relying on the CPU utilization results in a drastic degradation of the elastic deviation if the workload mix decreases. The optimization strives to an upper threshold, which approximates the CPU utilization in a fully utilized scenario. If the service is now fully utilized, but the

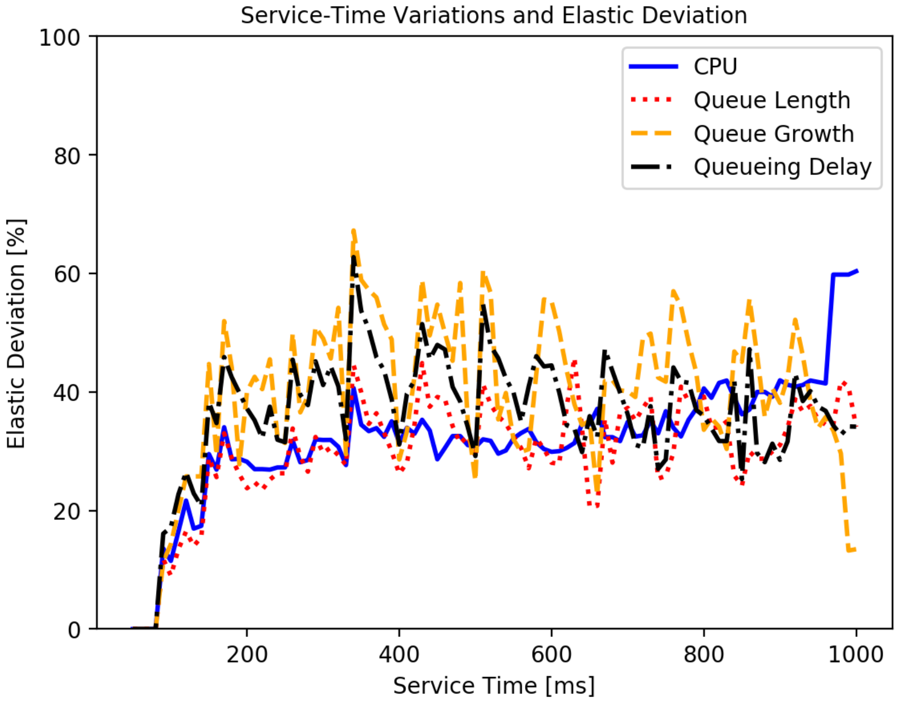


Figure 8.8.: Effect of service time variations on the elastic deviation.

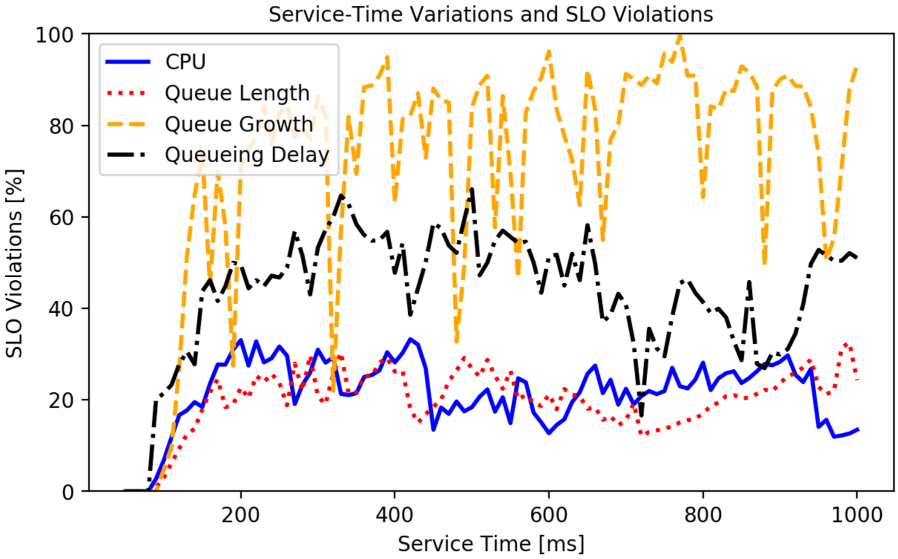


Figure 8.9.: Effect of service time variations on the SLO.

CPU share of the workload decreases, the upper threshold is not breached, resulting in no scaling decision at all. The elastic deviation steadily increases for an increasing CPU utilization. This is based on an increased overprovisioned timeshare $timeshare_o$ since the scaler provisions too aggressively.

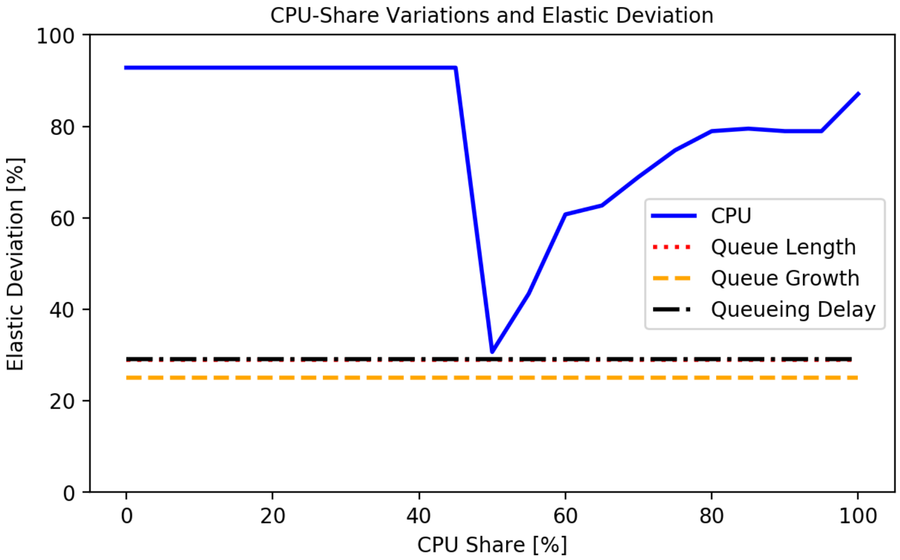


Figure 8.10.: Effect of CPU share variations on the elastic deviation.

Whereas the percentage of SLO violations stays constant for all queue metrics, it increases greatly for a decreasing CPU share and decreases for an increasing CPU share. Whereas the first is based on no scaling decisions, since the upper threshold is never breached, the last is based on the aggressive provisioning of the scaler, since the upper threshold is comparatively low for higher CPU share scenarios.

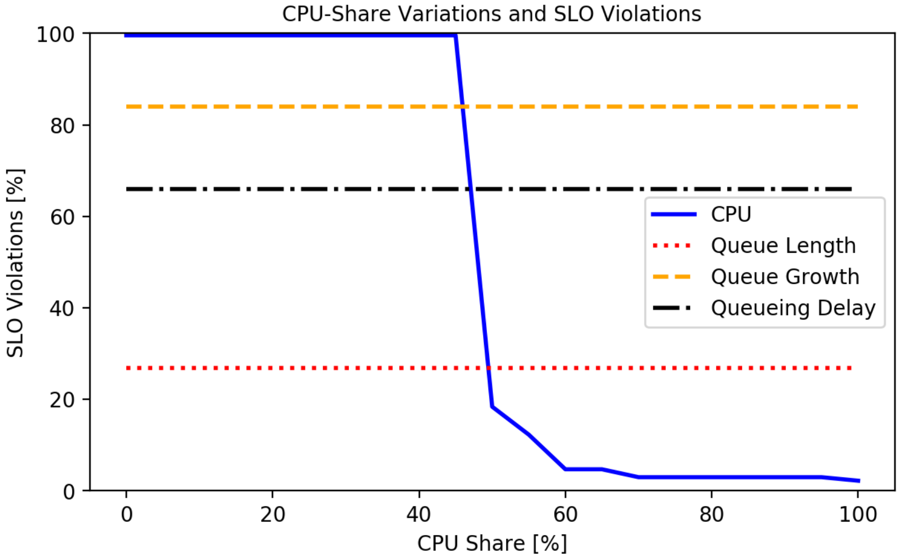


Figure 8.11.: Effect of CPU share variations on the SLO.

We conclude this experiment by emphasizing the low robustness of the CPU-based auto-scalers to cope with variations in the CPU share of the workload.

| | $\delta SLO$ (ST Var.) | $\delta\sigma$ (ST Var.) | $\delta SLO$ (CPU Var.) | $\delta\sigma$ (CPU Var.) |
|---|---|---|---|---|
| Queue Length | 6.7 | 10.6 | 4.7 | 34.6 |
| Queueing Delay | 8.4 | 15.3 | 4.2 | 20.4 |
| Queue Growth | 30.1 | 117.7 | 12.8 | 49.8 |
| CPU Utilization | 18.2 | 14.0 | 69.2 | 35.3 |

Table 8.2.: Prediction error of the simulation.

### 8.4.5. Q.1.4 – Simulation Model Accuracy

For each performance metric, we conduct controlled experiments for $L_{mix} = \{25, 50, 75, 100\}$ and $ST = \{250, 500, 750, 1000\}$. Table 8.2 summarizes the prediction error of SLO violations $\delta SLO$ and of the elastic deviation $\delta\sigma$.

The average prediction error for elasticity and SLO violations for an auto-scaler based on the queue length is 14.17 %, for the queueing delay 12.12 %, for the queue growth 52.64 % and for the CPU utilization 34.18 %. Since the simulation framework models the monitoring infrastructure, errors in modeling measurements of the arrival and departure rate cumulate in a high queue growth prediction error. The queueing delay has a low prediction error. Overall, the results of the controlled experiments show a satisfying prediction accuracy and support the findings of this evaluation.

### 8.4.6. Threats to Validity

We rely on the guidelines described in [63] to discuss threats to validity of the evaluation. A special focus lies on the experimental setup, which may affect the generalizability of the results.

**Internal validity.** We evaluate the impact of resource demand and capacity variations on scaling decisions. By varying the wait and computation time in a controlled manner, we express variations, which result in a change in the service time or CPU share. By validating an analytical and simulation model, we exclude the possibility of another influential factor affecting the service time or the CPU share. However, if the payload of messages is drastically increased or the message service time is strongly reduced, we expect a stronger influence of the communication mechanisms on the results, which may require refinements of the analytical model.

**External validity.** The microservice architectural style aims for loosely coupled and highly specialized microservices. Both, an asynchronous communication via message queues and a synchronous communication via the REST paradigm is common. Therefore we deem the message queues and the usage of their metrics for scaling decisions as relevant for service operators. The synthetic microservice is a simplified model of a real-world application for the SaaS use case. However, the fundamental architecture for a similar use case is discussed in [16]. For this reason we assume that it is sufficient to represent this class of cloud applications. We generalize the CPU results for microservices which do not utilize a message queue.

Threshold-based rules auto-scaling is one of the simplest mechanisms to provision resources in a cloud environment. In this setup we identified that the quality of performance metrics for scaling decisions depends on microservice characteristics. We cannot exclude the possibility that other scaling mechanisms are more adaptive to changes in the microservice characteristics and their impact on the performance metrics. However, since we perceive threshold-based rules as one of the most common scaling strategies in industry we see validity of our evaluation in real-world scenarios. By creating an analytical model of the influence of the external service characteristics on the microservices CPU utilization and throughput we emphasize the validity of the case study in an analytical manner.

**Construct validity.** We derive the robustness of each setup by comparing the elasticity achieved of a auto-scaler in different scenarios. By capturing timing and accuracy aspects they provide an application-agnostic measure to evaluate the performance of auto-scalers. However, a service operator may have multiple goals which include SLO conformance. In these cases the elasticity measures can be weighted or combined with SLO violations. We assume, that the general findings of the evaluation are still valid in these cases.

**Reliability validity.** By describing the analytical and simulation model and the experimental setup we strongly assume that the results are reproducible thus making it possible for another researcher to conduct the same evaluation and obtain the same or very similar results. We see a limitation in the dynamic nature of the cloud environment which could possible change the optimal thresholds based on the time of experiment.

### 8.4.7. Discussion

In this section we have answered RQ 1. We validated an analytical and simulation model by conducting measurements. We have shown that thresholds based on message queue metrics are robust towards resource demand and capacity variations. The elasticity of CPU-based auto-scalers is greatly decreased if the CPU share of processing messages decreases, e.g. due to an increased wait time or computation capacity. We have shown, that these result in drastic changes of the SLO conformance for the CPU utilization. Otherwise, the CPU utilization offers a low elastic deviation and a high SLO conformance. For this reason we highly recommend a queue-based auto-scaler for runtime managing microservices with a high variation in the computation capacity or wait time. For some microservices it may be sufficient to configure a comparatively low upper threshold in order to be more robust to CPU share variations. However, this probably results in a lesser degree of elasticity by overprovisioning, which affect the operating costs.

## 8.5. Congestion Avoidance Characteristics of time-driven Flow Control

This section investigates the congestion avoidance characteristics of the time-driven flow control approaches presented in chapter 5. Therefore, it addresses Goal 2 of the validation.

In the congestion-avoidance mode the approaches aim to fully utilize the available capacity by continuously probing the capacity with transmission rate adjustments. We analyze varying load and capacity scenarios, in which we adjust the number of connected devices or the cloud service capacity during runtime. Since sensing cloud applications differ in timeliness demands, we create a pareto curve for each scenario. The pareto curve captures the congestion avoidance efficiency in terms of the average achieved service utilization and the congestion-induced processing delay. In order to construct the curves, we search for optimal configurations for a range of target service utilizations aiming for a minimal congestion-induced delay. The pareto curve can be used to select a strategy and configure it to reach a target service utilization based on the application-specific trade-off between the achieved utilization and the congestion-induced delay. Therefore, we do not introduce QoS cost functions to quantify timeliness or accuracy costs.

### 8.5.1. Experimental Design

The goal of the experiment is to retrieve the characteristics of each flow control strategy in terms of the congestion avoidance efficiency. The congestion avoidance efficiency is expressed by a curve, which consists of the average utilization over a minimal congestion-induced delay. In order to quantify the service utilization we rely on the average CPU utilization of the microservice as a proxy metric. Furthermore, we use the measured queueing delay as a proxy for the congestion-induced delay.

Retrieving such curves is computational expensive since the mapping between configuration and resulting service utilization and queueing delay is complex. Instead of brute forcing configuration candidates, we instantiate the optimization framework and simulation model as described in section 8.2.2. By relying on DE, we are able to reduce the search effort by heuristically creating candidates for the pareto curve input. We configure the simulation model based on the ShapeShifter case study characteristics. These characteristics include the monitoring policies of the measurement providers and the reconfiguration delay of smart devices. Finally, we validate the findings and determine the simulation prediction accuracy by conducting experiments on the deployed case study system on the Bosch IoT Cloud.

**Flow Control Candidates.** We evaluate the following flow control strategies: capacity-estimating flow control (CEF) and the TCP-inspired flow control (TIF) with four different control schemes: AIMD, MIMD, AIAD, MIAD. We also include the TIF approaches with load extension (TIF-L), to evaluate its impact on the adaptation behavior, especially in varying connectivity scenarios.

**Capacity and Connectivity Scenarios.** Figure 8.12 shows the capacity and connectivity scenarios. Based on the objective, we either adjust the capacity or the connectivity during the experiment. The steady scenario aims to observe the steady state behavior of each strategy. The dynamic scenarios with an increasing and decreasing capacity or connectivity aims to capture the quality of adapting to changes. In order to allow a fair comparison each scenario has an average processing capacity of $c = 22\frac{\text{msg}}{\text{sec}}$ and an average number of 22 connected devices.

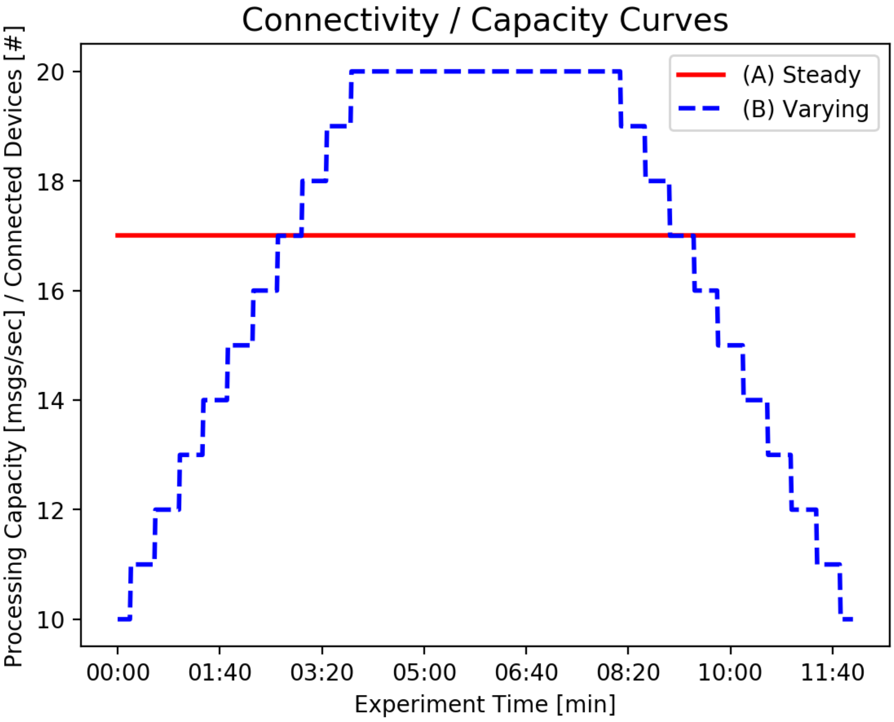


Figure 8.12.: Connectivity and capacity variation scenarios to investigate the impact on the congestion-avoiding flow control strategies.

**Configuration Space.** Each flow control approach utilizes an increase and decrease parameter. The parameters affect the congestion avoidance behavior to a great extent. Since all approaches rely on a binary congestion feedback, we provide congestion observers with the same configuration. The approaches consist of the following parameter tuples $K$, which are reconfigured during the candidate search:

- **CEF:** Uses an overload protection multiplicator $k_{protect}$ to accelerate the congestion recovery and a recovery multiplicator $k_{recover}$ to refine the current capacity estimation: $K_{CEF} = (k_{protect}, k_{recover})$

- **TIF:** Uses an increase and a decrease parameter, which is, based on the scheme, of additive or multiplicative nature: $K_{TIF} = (k_{increase}, k_{decrease})$

**Congestion Observer.** We configure the congestion observer to monitor the queue length with a moving average size of 10 and a congestion threshold of 10. The congestion observer retrieves every second an update from the infrastructure.

**Pareto Curve Creation.** To create the pareto curve, we introduce utilization slots of the size of 5% in the range of $0 - 100\%$. For each slot, we search for the best configuration in terms of the minimal queueing delay. We rely on an experience-based iteration depth of 5000 and an initial population size of 40.

To evaluate the fitness of each configuration candidate, we rely on the absolute difference in the service utilization and the measured queueing delay. We calculate the absolute deviation with $\Delta U = |U_{is} - U_{slot}|$, based on the target service utilization $U_{slot}$ and the measured service utilization $U_{is}$. The fitness function $F$ weights the service utilization deviation to a greater extend but also considers the queueing delay $T_{QDelay}$:

$$F = \Delta U^2 + \Delta U * T_{QDelay}$$

However, instead of using the best candidates only, we include each evaluated candidate as an input for the pareto curve, if its achieved queueing delay is below the lowest queueing delay of the corresponding utilization slot. Furthermore, we discard measurements which have resulted in a queueing delay of 0 since it indicates, that the configuration of the strategy has not led to a congestion avoidance cycle during the experiment.

**Pareto Curve Analysis.** In order to attribute to qualitative differences across the overall utilization range, we split the pareto curve in four slots of different sizes. The first slot contains a low utilization range of $0 - 50\%$, in which we expect the approaches to behave similar. To analyze the behavior in a higher utilization range, we introduce more fine-granular slots: $50 - 78\%$, $70 - 85\%$ and $85 - 100\%$. We capture for each slot the average queueing delay. In order to compare the flow control strategies we also capture the overall average and median queueing delay across all slots. Furthermore, we compare the combined results of TIF with the combined results of TIF-L in order to express qualitative differences between both.

**Adaptation Protection Time.** The adaptation protection time of each strategy is $\tau_{Adaptation} = 5\,\text{s}$. Between an adaptation protection interval, no reconfiguration decision takes place. The time is based on the reactivity of the congestion observer, which relies on a message queue measurement provider. The message broker used in the ShapeShifter case study is a RabbitMQ broker, which monitors with an interval of 5 s. Therefore, we assume, that the adaptation protection time is sufficient to observe an effect of a reconfiguration decision in the subsequent adaptation step.

**Initial Transmission Rate.** In each scenario, we set the initial transmission rate of each created device to $T_{init} = 1\frac{\text{msg}}{\text{sec}}$. We set the initial control scheme state of the TIF approaches to $T_{TIF} = T_{init}$. To allow a fair comparison with the load extension, we multiply the initial transmission rate with the number of connected devices $N_{Devices}$, such that $T_{TIF-L} = N_{Devices} * T_{init}$.

**Experiment Setup.** We set the CPU share of messages to 100 % in order to have a proxy metric of the service utilization. Furthermore, we set the latency distribution to be deterministic at 0 s. The overall duration of the experimental run is $t_{experiment} = 12\,\text{min}$.

### 8.5.2. Q.2.1 – Congestion Avoidance Efficiency in a Steady Capacity and Connectivity Scenario

The pareto curves retrieved from the experiments are shown in figure 8.13 and summarized in table 8.3.

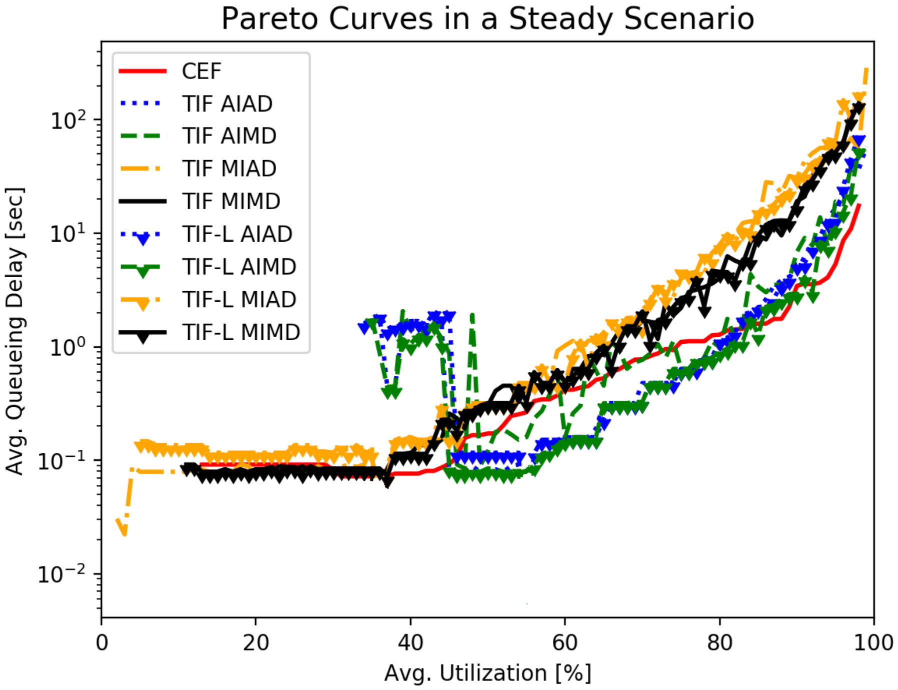


Figure 8.13.: Pareto curve of the average utilization and queueing delay in a steady capacity and connectivity scenario.

For utilization ranges below 50 % the strategies result in a similar utilization to queueing delay ratio. However, TIF approaches based on a AIAD schemes result in slightly increased queueing delays. Within an average utilization of 50 % – 70 % the characteristics of the strategies start to spread apart, especially for control schemes relying on a multiplicative increase. In an upper utilization range of above 70 % the quality of the strategies variates greatly. In this range, TIF AIMD and TIF-L AIMD achieve with 1.27 s and 0.95 s the lowest average queueing delay, closely followed by the CEF approach with 1.33 s. In utilization scenarios above 85 % the queueing delay increases drastically for all approaches. Notably, CEF approach maintains a comparatively low queueing delay of 5.65 s, followed by TIF AIMD and TIF-L AIMD with 17.17 s and 10.87 s.

Based on the adaptation interval and the moving average, the congestion observer perceives congestion delayed. Therefore, the control scheme may decide to further increase the transmission rate. This can be observed for TIF request schemes with a multiplicative in target utilization above 50 %. Since the transmission rate is increased multiplicative, the adaptation decision impacts the resulting load to a great extent. Whereas it achieves a high service utilization, it accumulated messages in the queue, resulting in a comparatively high queueing delay. Approaches with an additive increase adjust in a linear manner, which

| Strategy | Queueing Delay [sec] | | | | | |
|---|---|---|---|---|---|---|
| | Average | Median | 0-50 [%] | 50-70 [%] | 70-85 [%] | 85-100 [%] |
| CEF | 1.33 | 0.52 | 0.25 | 0.56 | 1.33 | 5.65 |
| TIF AIAD | 5.08 | 0.76 | 1.01 | 0.26 | 1.5 | 20.75 |
| TIF AIMD | 4.5 | 0.95 | 0.93 | 0.46 | 1.4 | 17.17 |
| TIF MIAD | 18.29 | 0.67 | 0.17 | 9.04 | 25.57 | 92.45 |
| TIF MIMD | 7.42 | 0.45 | 0.12 | 0.76 | 4.13 | 43.92 |
| TIF-L AIAD | 3.9 | 0.72 | 1.22 | 0.18 | 1.01 | 16.26 |
| TIF-L AIMD | 2.67 | 0.64 | 0.71 | 0.18 | 0.86 | 10.87 |
| TIF-L MIAD | 9.74 | 0.32 | 0.15 | 0.89 | 6.87 | 60.6 |
| TIF-L MIMD | 7.34 | 0.39 | 0.12 | 0.75 | 3.84 | 43.73 |
| TIF | 9.7 | 0.68 | 0.36 | 2.63 | 8.15 | 42.66 |
| TIF-L | 6.41 | 0.48 | 0.36 | 0.5 | 3.14 | 32.87 |

Table 8.3.: Results of the steady capacity and connectivity scenario.

allows to find candidates achieving a high service utilization and avoiding long-lasting congestions. The configuration of TIF(-L) and CEF is highly tunable in a steady scenario, since it converges to the capacity with an oscillation which amplitude is determined by the parametrization of the increase and decrease parameters. In theory, the load extension of TIF has no influence on the adaptation quality in a steady scenario and behaves equivalent to TIF. However, in the experiments we measure slight variations with an average deviation of 14 %.

Overall, the CEF offers a high congestion avoidance efficiency in a wide range of target utilizations. However, TIF or TIF-L with AIMD or AIAD are also very strong candidates and surpass CEF in utilization areas below 85 %, without inducing the same architectural restrictions. However, the CEF approach allows a higher utilization of the service without drastically increasing the queueing delay.

### 8.5.3. Q.2.2 – Congestion Avoidance Efficiency in a Varying Capacity Scenario

The pareto curves for a scenario with a varying cloud service capacity but a constant number of devices are shown in figure 8.14 and are summarized in table 8.4.

As in the steady connectivity and capacity scenario, the approaches have a similar characteristic for an average utilization below 50 %. For utilizations between 50 % – 85 %, TIF(-L) with AIMD and AIAD outperforms the other approaches, with an average queueing delay between 0.29 s – 2.97 s. In a high utilization range of 85 % – 100 % CEF surpasses the TIF approaches, with an average queueing delay of 8.51 s.

By varying the capacity, CEF is challenged to estimate the time-varying capacity with a high accuracy. In scenarios with an utilization below 85 % congestions are less severe. Since CEF estimates the capacity in congestion state, it is readjusted less often. We assume, that this contributes to the qualitative degradation compared to the previous experiment. Since the experiment considers a change in the capacity but not in the number of connected

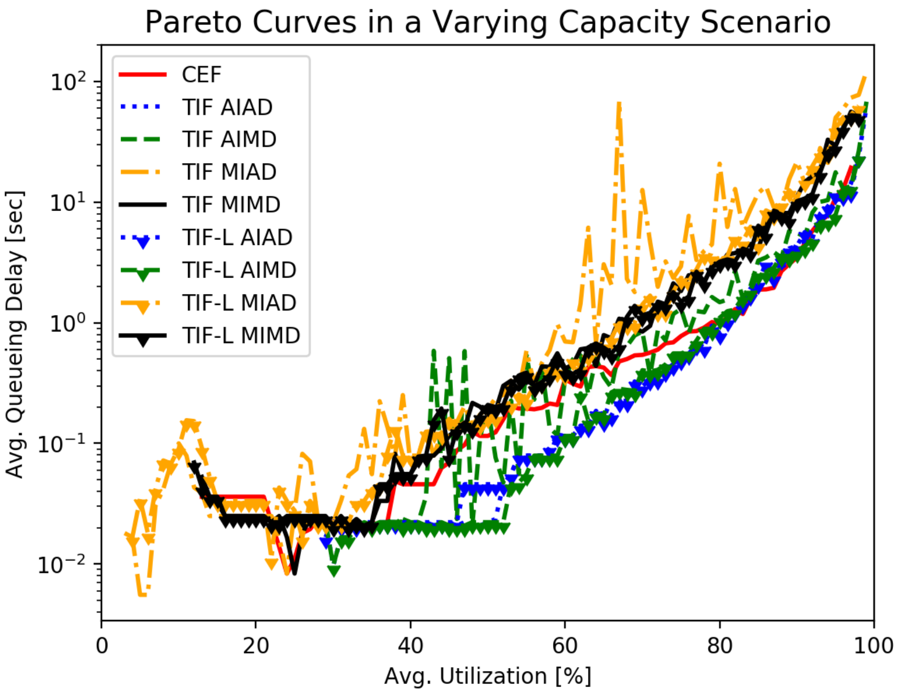


Figure 8.14.: Pareto curve of the average utilization and queueing delay in a varying capacity and steady connectivity scenario.

devices, the optimization of the configuration parameters should result in an equivalent behavior for both TIF approaches. However, based on the results, we measure an average deviation of 27 % in the achieved average queueing delay.

In this scenario, TIF(-L) AIMD and AIAD outperforms CEF slightly. TIF(-L) schemes based on multiplicative increase maintain an inferior congestion avoidance efficiency.

### 8.5.4. Q.2.3 – Congestion Avoidance Efficiency in a Varying Connectivity Scenario

Figure 8.15 shows the result for a varying connectivity scenario, whereas table 8.5 summarizes the queueing delay for each strategy across different utilization ranges.

Within a utilization range of 50 % – 70 % TIF(-L) AIMD and AIAD provide the highest congestion avoidance efficiency. They are closely followed by CEF. In the utilization range between 70 % – 85 % TIF-L AIMD and AIAD surpasses the approaches without load extension. In utilization scenarios above 70 % CEF offers with an average of 6.41 s a low queueing delay but is overall on par with TIF-L AIMD and AIAD, which achieve 7.25 s, respectively 6.01 s.

The load extension of TIF allows to react to connectivity change in a proportional manner. That means, that adaptations result in the same effect on the overall load independently of the number of connected devices. Therefore, it offers an average improvement of the queueing delay of 200 % compared to TIF without load extension. The CEF approach

| Strategy | Queueing Delay [sec] | | | | | |
|---|---|---|---|---|---|---|
| | Average | Median | 0-50 [%] | 50-70 [%] | 70-85 [%] | 85-100 [%] |
| CEF | 1.78 | 0.5 | 0.19 | 0.57 | 1.62 | 8.51 |
| TIF AIAD | 11.66 | 1.29 | 0.56 | 0.41 | 2.97 | 44.98 |
| TIF AIMD | 6.14 | 1.15 | 0.71 | 0.49 | 2.41 | 22.47 |
| TIF MIAD | 31.47 | 1.88 | 0.96 | 21.38 | 54.83 | 141.13 |
| TIF MIMD | 8.59 | 0.54 | 0.13 | 1.1 | 6.1 | 53.06 |
| TIF-L AIAD | 3.52 | 0.72 | 0.29 | 0.23 | 1.61 | 13.99 |
| TIF-L AIMD | 3.2 | 0.66 | 0.34 | 0.25 | 1.42 | 13.1 |
| TIF-L MIAD | 10.87 | 1.17 | 0.24 | 2.8 | 11.74 | 63.95 |
| TIF-L MIMD | 7.17 | 0.73 | 0.16 | 1.26 | 6.34 | 41.43 |
| TIF | 15.9 | 0.98 | 0.59 | 5.85 | 16.58 | 62.97 |
| TIF-L | 6.75 | 0.73 | 0.23 | 1.14 | 5.28 | 32.33 |

Table 8.4.: Results of the varying capacity and steady connectivity scenario.

| Strategy | Queueing Delay [sec] | | | | | |
|---|---|---|---|---|---|---|
| | Average | Median | 0-50 [%] | 50-70 [%] | 70-85 [%] | 85-100 [%] |
| CEF | 1.0 | 0.09 | 0.02 | 0.28 | 1.3 | 6.41 |
| TIF AIAD | 2.46 | 0.04 | 0.0 | 0.1 | 1.79 | 15.57 |
| TIF AIMD | 1.66 | 0.02 | 0.01 | 0.1 | 1.51 | 10.78 |
| TIF MIAD | 12.46 | 0.11 | 0.03 | 1.3 | 18.21 | 94.51 |
| TIF MIMD | 5.81 | 0.19 | 0.03 | 0.44 | 5.72 | 42.76 |
| TIF-L AIAD | 1.06 | 0.04 | 0.0 | 0.04 | 0.53 | 6.01 |
| TIF-L AIMD | 1.24 | 0.05 | 0.0 | 0.04 | 0.48 | 7.25 |
| TIF-L MIAD | 4.44 | 0.05 | 0.01 | 0.72 | 6.05 | 33.77 |
| TIF-L MIMD | 4.42 | 0.05 | 0.01 | 0.68 | 5.93 | 29.5 |
| TIF | 6.05 | 0.05 | 0.02 | 0.48 | 6.81 | 40.91 |
| TIF-L | 3.1 | 0.05 | 0.01 | 0.37 | 3.25 | 19.13 |

Table 8.5.: Results of the steady capacity and varying connectivity scenario.

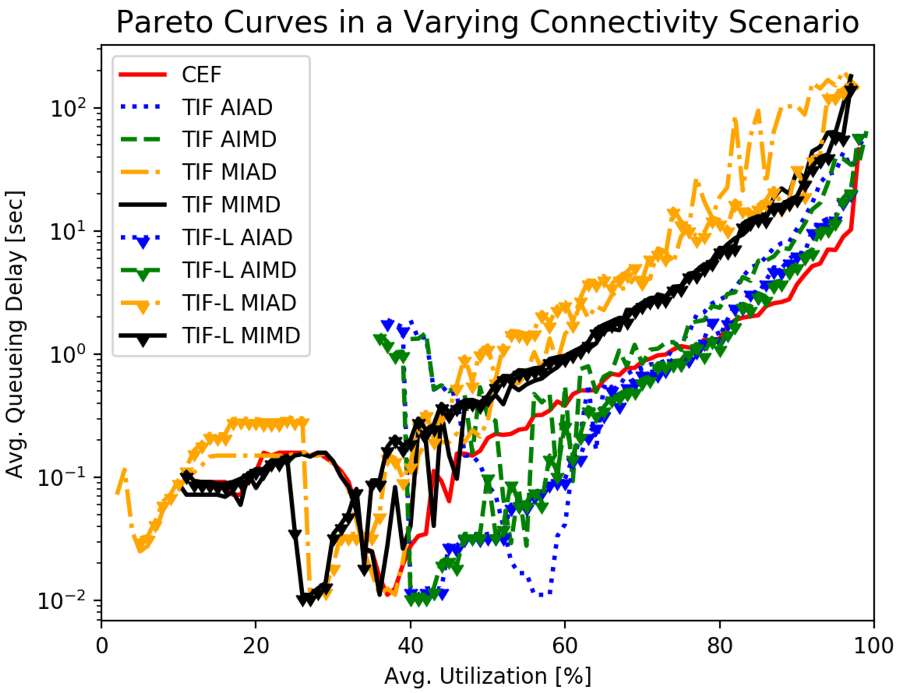


Figure 8.15.: Pareto curve of the average utilization and queueing delay in a varying load scenario.

performs well, but exhibits slight degradations compared to TIF-L approaches with AIMD or AIAD.

The scenario has shown, that the load extension of TIF results in improvements in scenarios with a varying capacity. The CEF approach maintains a high congestion avoiding efficiency across all target utilizations.

### 8.5.5. Q.2.4 – Simulation Model Accuracy

For each approach in each scenario $S = \{Steady, VaryingCapacity, VaryingConnectivity\}$ we conduct 4 measurements, resulting in a total of 81 measurements to evaluate the accuracy of the simulation model. We evaluate the following target utilizations $U = \{60, 70, 80, 90\}$ in order to capture the accuracy in utilization ranges with significant different congestion characteristics. For each scenario and strategy, we determine the average relative prediction error as percentage of the CPU utilization $\delta CPU$ — which acts as a proxy for the service utilization — and of the average queue length $\delta Queue_{Length}$ and queueing delay $\delta Queue_{Delay}$. Furthermore $\sigma Queue_{Delay}$ denotes the median queueing delay prediction error.

**Steady.** Table 8.6 summarizes the prediction accuracy for of each strategy in a steady connectivity and capacity scenario.

Overall, the simulative predictions are accurate, with an average prediction error of 8 % of the CPU utilization and a median prediction error of 31.1 % for the queueing delay.

| Strategy | $\delta CPU$ | $\delta Queue_{Length}$ | $\delta Queue_{Delay}$ | $\sigma Queue_{Delay}$ |
|---|---|---|---|---|
| CEF | 3.9 | 45.9 | 107.1 | 72.9 |
| TIF AIAD | 0.0 | 52.0 | 41.9 | 48.4 |
| TIF AIMD | 2.7 | 106.6 | 195.0 | 118.1 |
| TIF MIAD | 19.5 | 58.1 | 39.1 | 34.5 |
| TIF MIMD | 3.7 | 38.3 | 22.1 | 24.7 |
| TIF-L AIAD | 7.7 | 46.6 | 18.3 | 20.7 |
| TIF-L AIMD | 6.9 | 23.8 | 43.5 | 29.9 |
| TIF-L MIAD | 12.2 | 55.1 | 34.4 | 25.0 |
| TIF-L MIMD | 6.7 | 33.0 | 47.0 | 51.6 |
| Average | 7.8 | 51.0 | 60.9 | 34.5 |

Table 8.6.: Average and median prediction errors as percentage difference for the service utilization, queue length and queueing delay in a steady scenario.

| Strategy | $\delta CPU$ | $\delta Queue_{Length}$ | $\delta Queue_{Delay}$ | $\sigma Queue_{Delay}$ |
|---|---|---|---|---|
| CEF | 4.8 | 17.4 | 52.7 | 44.6 |
| TIF AIAD | 11.8 | 47.6 | 36.4 | 36.4 |
| TIF AIMD | 1.6 | 49.8 | 129.0 | 173.7 |
| TIF MIAD | 41.1 | 55.8 | 51.2 | 62.6 |
| TIF MIMD | 2.9 | 25.8 | 76.1 | 30.9 |
| TIF-L AIAD | 6.8 | 17.1 | 126.7 | 71.2 |
| TIF-L AIMD | 1.3 | 27.2 | 171.8 | 118.2 |
| TIF-L MIAD | 20.0 | 51.8 | 55.2 | 56.4 |
| TIF-L MIMD | 2.1 | 22.8 | 43.2 | 17.7 |
| Average | 11.3 | 35.0 | 82.5 | 62.6 |

Table 8.7.: Average and median prediction errors as percentage difference for the service utilization, queue length and queueing delay in a varying capacity scenario.

The high accuracy in predicting the CPU utilization shows, that the simulation is able to precisely simulate the congestion avoidance dynamics. The simulation is less accurate in predicting the queueing delay or queue length, but still produces decent predictions. The simulation aims to reproduce the measurements provided by RabbitMQ. It performs well for low utilization ranges, but underestimates the queue length and queueing delay for an utilization of $\geq$ 90 % resulting in a high prediction error. The high prediction error on this target utilization is reflected in the comparatively high average queueing delay but low median queueing delay.

**Varying Capacity.** Table 8.7 summarizes the prediction accuracy for of each strategy in a varying capacity scenario.

The prediction error stays low in predicting the service utilization with 10.7 % but increases greatly in predicting the queueing delay, which exhibits a median prediction error of 69.1 %. The simulation underestimates the queueing delay and queue length in high utilization ranges. Overall, the prediction is degraded in this scenario, based on

| Strategy | $\delta CPU$ | $\delta Queue_{Length}$ | $\delta Queue_{Delay}$ | $\sigma Queue_{Delay}$ |
|---|---|---|---|---|
| CEF | 2.7 | 38.0 | 71.9 | 64.2 |
| TIF AIAD | 2.5 | 42.1 | 48.1 | 32.8 |
| TIF AIMD | 0.2 | 38.7 | 16.5 | 10.7 |
| TIF MIAD | 12.9 | 59.5 | 49.8 | 53.5 |
| TIF MIMD | 9.0 | 21.3 | 20.7 | 21.7 |
| TIF-L AIAD | 3.9 | 47.2 | 43.9 | 29.5 |
| TIF-L AIMD | 1.3 | 35.3 | 13.9 | 14.4 |
| TIF-L MIAD | 10.7 | 55.4 | 44.2 | 42.7 |
| TIF-L MIMD | 2.7 | 26.1 | 20.0 | 11.1 |
| Average | 5.1 | 40.4 | 36.6 | 29.5 |

Table 8.8.: Average and median prediction errors as percentage difference for the service utilization, queue length and queueing delay in a varying connectivity scenario.

the technical realization of the capacity variations. The ShapeShifter service is adjusted every second by a controller managed by SEIA. Based on the deployment on a distributed infrastructure, the reconfiguration decision may experience delays. This challenges the simulation, since it assumes an instant capacity adjustment.

**Varying Connectivity.** Table 8.8 summarizes the prediction accuracy for of each strategy in a steady connectivity and capacity scenario.

The simulation accuracy for a varying connectivity scenario is very high, with an average prediction error of 5.2 % for the utilization and 28.0 % for the queueing delay. Overall, the simulation correctly simulates the dynamics induced by changing devices. Since smart devices are synthetic entities in the ShapeShifter case study, there are no adjustment dynamics compared to a distributed setup.

**Discussion.** Overall, we deem the simulation predictions as accurate enough to analyze scenarios experienced by a single service Cloud-IoT application. The simulation is very precise in predicting the service utilization but less in the queueing delay. The prediction degrades for queue metrics in high utilization scenarios.

### 8.5.6. Threats to Validity

We rely on the guidelines described in [63] to discuss threats to validity of the evaluation. A special focus lies on the experimental setup, which may affect the generalizability of the results.

**Internal validity.** The presented approaches reconfigure the transmission rate of smart devices to utilize the capacity of a cloud service. In order to converge to a specific utilization the approaches induce short-term congestions. Since we operate both the message broker and the message queue in isolation and the measured arrival rate of the queue is in line with the expected rate of produced messages by the smart devices, we exclude an interfering factor and strongly assume a causal relation between transmission rate adaptations and congestion avoidance.

**External validity.** The running example is based on a typical architecture for sensing Cloud-IoT solutions. Since smart devices are operated as virtual entities, which are deployed as microservices on the cloud infrastructure, there are some factors threatening the generalization of the results. One of these factor is a nearly uniform network latency to each virtual smart device, which we can expect to be not the case in a productive setup. In general, we expect a higher and more varying network latency in real world scenarios. We expect these latencies to degrade the adaptation quality by decreasing the responsiveness of the approach. However, we expect the qualitative results of this work as generalizable for similar application setups. Furthermore, the synthetic microservice is a simplified model of a real-world application. However, on the one side, some Cloud-IoT applications consist of a single processing microservice storing messages in a database, and on the other side, we investigate the flow control behavior in respect to the induced load and available capacity, where it does not matter, how many services contribute to the overall capacity. Therefore, we deem the results as generalizable.

**Construct validity.** In our setup we analyze message queue metrics to derive the degree of congestion avoidance. By communicating via a message queue, an imbalance between the cloud services' capacity and the overall load on it results in an accumulation of messages in the queue. For this reason, we rely on message queue metrics, in order to express the congestion-induced delay and to recognize congestions by observing the queue length.

**Reliability validity.** To the best of our knowledge, we provided all details needed to replicate the experimental setup. For this reason, we strongly expect the results to be reproducible.

### 8.5.7. Discussion

In this set of experiments, we have investigated the congestion avoidance characteristics of the set of flow control strategies.

Overall, the CEF approach provides a high congestion efficiency in all scenarios. It performs well in a steady and in a varying load scenario and shows slight adaptation quality degradations in a varying capacity scenario. It is closely followed by TIF(-L) approaches with AIMD or AIAD request schemes, which in contrast to CEF do not impose architectural restrictions. In some cases, they outperform CEF. In varying connectivity scenarios, TIF approaches without load extension are inferior, since the resulting shift in the load based on an additive increase or decrease adjustment depends on the number of active devices.

The pareto curves can be used to compare the approaches in terms of their service utilization and congestion characteristics. They influence the QoS of sensing cloud applications to a great extent, since the achieved service utilization determines the average transmission interval in congestions, which influences the accuracy. Congestion-induced delays affect the timeliness. Therefore, a suitable trade-off should be met by a service operator.

## 8.6. QoS Characteristics of time-driven Flow Control

This section targets Goal 3 of this thesis by investigating the QoS characteristics of time-driven flow control approaches in overload-protection mode. In contrast to the previous investigation the approaches enforce an upper limit for the transmission rate at which the cloud applications accuracy demand is considered to be satisfied. We compare these approaches to auto-scalers which provide a complementary mechanism by resource provisioning. Based on normalized QoS cost functions we investigate the composition of the QoS costs of each approach in intensifying overload scenarios. To enable a fair comparison between the approaches, we optimize the configuration of each approach in each overload scenario. Furthermore, as stated in Goal 4, we investigate the QoS characteristics of coupling mechanisms which combine time-driven flow control with auto-scaling as presented in chapter 6. Whereas the results are obtained by simulation, we obduct controlled experiments on the Bosch IoT Cloud to address Goal 5.

### 8.6.1. Experimental Design

In order to capture the impact of overload situations and reconfiguration decisions on the QoS conformance we rely on accuracy, timeliness and resource cost functions, as introduced in chapter 4. We express the QoS characteristics of each approach in intensifying overload scenarios by the composition of these costs. We compare flow control approaches with auto-scaling using a static provisioned cloud service as baseline. In each scenario, we normalize the QoS cost functions based on the expected performance of a theoretical optimal auto-scaler or flow control approach. Furthermore we optimize the configuration of each approach in each overload scenario using `SEIA` to instantiate the optimization framework described in section 8.2.2.

**Case Study.** We rely on the ShapeShifter case study, since it provides a simple but complete architecture for a sensing cloud application.

**Flow Control Candidates.** We select the following flow control approaches, since they have shown a high congestion avoidance efficiency in the investigation conducted in validation chapter 8.5:

- **TIF AIMD.** The approach utilizies an AIMD scheme in order to adjust the transmission rate of all smart devices. It has shown a high congestion avoidance efficiency but a vulnerability to connectivity variations.
- **TIF-L AIMD.** This approach utilizes a load extension, in which the overall load is adjusted according to an AIMD scheme. It has shown a high congestion avoidance efficiency by sharing the load fairly across the current number of active devices.

- **CEF.** As a main refinement to the TIF approaches, it utilizes a capacity-estimation module. Therefore, it aims to reduce the time needed to find a congestion avoiding transmission rate. It has shown a high congestion avoidance efficiency, especially for high service utilizations.

**Auto-Scaler Candidates.** Many auto-scaling approaches have been proposed in literature [49]. We conduct experiments in the range of minutes, which aim to be efficient in terms of observable effects and duration. Therefore, we focus on auto-scalers which rely on a recent history to make predictions [42][21] in contrast to auto-scalers using long-term history data [71] or being hybrid in nature [7]. We select the following auto-scaler candidates:

- **React.** Presented in [21], React is a mechanism which provisioning decisions are based on thresholds. It also introduces a cooldown duration in order to avoid oscillating scaling decisions. The scaling mechanism is realized as a built-in scaler on many cloud platforms and is widely used in industry labeled as threshold-based rules auto-scaler. Scaling decisions are usually based on the CPU utilization.

- **Reg.** We select a regression-based auto-scaler (Reg) [42] as state of the art candidate. Whereas it reacts to load by scale-up decisions, it uses a regression-based prediction for scale-down decisions, when the capacity exceeds the load. As in [7] we configure the regression window size to 60 data points, which translates to 60 s. We obtain the implementation from [56], which relies on the capacity per instance and the arrival rate in order to calculate the number of required service instances. Since the approach aims to scale web services, it relies on proxy logs of $t$ seconds to determine the 95th percentile of the average response time. We refine this mechanism to rely on a capacity estimation based on message queue metrics as presented in section 5.4.1. Overall, the main difference to React is the ability to scale multiple instances at once, based on the (estimated) arrival rate and the capacity per instance.

**Coupling Candidates.** Based on the coupling mechanisms presented in chapter 6, we investigate the following candidates:

- **Concurrent Coupling.** Described in section 6.3, this approach operates strategies concurrently without any orchestration resulting in an emergent behavior.

- **QoS Rules-Based Coupling.** Presented in section 6.4.4, the approach utilizes a set of threshold-based activation rules to decide, when to active a strategy on the basis of QoS cost functions.

- **Fuzzy-Rules Coupling.** In contrast to the rule-based coupling mechanisms, the Fuzzy-Rules based coupling approach presented in 6.5 aims to provide a comprehensible rule set to express reusable and complex rules.

**Flow Control and Auto-Scaler Combinations.** Each coupling mechanism offers an auto-scaling system binding, which (de-)activates all deployed auto-scaling systems of a cloud application. We investigate the coupling in a single service cloud application, therefore we combine a flow control and an auto-scaling strategy. Each coupling approach is investigated by relying on each combination of the flow control and auto-scaling strategies. The goal is to capture the emergent behavior of different mechanisms and observe the impact on the QoS characteristics of each approach. We select $S_{flow-control} = \{CEF, TIFAIMD\}$ as flow control approaches and $S_{auto-scaling} = \{React(CPU), Reg\}$ as auto-scaling approaches, resulting in overall of 4 combinations. We exclude the TIF-L AIMD approach, since it exhibits a similar and slightly inferior behavior to the CEF approach in high utilization ranges as concluded in the congestion avoidance investigation.

**Overload Scenarios.** We characterize overload situations in a time and intensity dimension. We quantify overload situations relying on the supported number of devices by the current service capacity $c$. We consider a device supported by the given capacity, if it is able to transmit data with a target transmission rate $T_{tp}$, at which the data accuracy is saturated, without experience timeliness degradations based on congestions. Therefore, we calculate the number of supported devices $N_{supported}$ as follows:

$$N_{supported} = \left\lfloor \frac{N_{instances} * c}{T_{tp}} \right\rfloor$$

We consider the application as overloaded, if the number of connected devices $N_{is}$ exceeds the number of supported devices, such that $O(t) = max(min(N_{is}(t) - N_{supported}, 1), 0)$. We do not investigate scenarios, in which the application is not overloaded. Therefore, the time share of an overload situation within an experiment of duration $T$ can be calculated as follows:

$$overload_{timeshare} = \frac{\sum_1^T O(t)dt}{T}$$

The intensity describes the severity of the overload situation in terms of the average ratio of the required to the supported devices:

$$overload_{intensity} = \frac{\sum_1^T (O(t) * \frac{N_{is}(t)}{N_{supported}})dt}{\sum_1^T O(t)dt}$$

Figure 8.16 illustrates the approach.

We introduce a measure, which combines both dimensions in order to quantify overload severities:

$$overload_{measure} = \frac{overload_{intensity} + overload_{timeshare}}{2}$$

**Cost Functions.** We instantiate the QoS functions introduced in section 4.4 using an input metric $k$, a target point $k_{tp}$ and a worst point $k_{wp}$. Based on these, we compute the QoS costs using a linear error function. We focus on accuracy, timeliness and resource cost

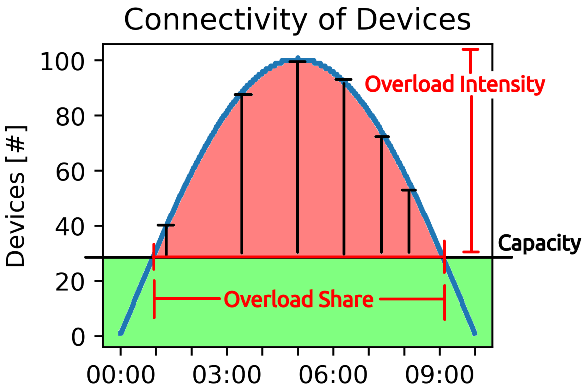


Figure 8.16.: Illustration of the overload quantification. The overload intensity is based on the deviation between supported and connected devices, whereas the overall time spent in an overload state is quantified by the overload share.

.

functions and provide for each input metrics. This approach assumes a linear relation between measured input and resulting QoS costs, which may not be the case in practice. We construct each cost function as follows:

- **Accuracy:** We quantify accuracy costs based on the average transmission rate $T$ as an inverse of the collection interval during the experiment. We associate no accuracy costs, if $T$ is above the target point $T_{tp} = 1\frac{\text{msg}}{\text{sec}}$. Therefore, we use the following cost function:

$$Q_{Accuracy}(T) = \begin{cases} \dfrac{T_{tp} - T}{T_{tp} - T_{wp}}, & \text{if } T \leq T_{tp} \\ 0, & \text{otherwise} \end{cases}$$

- **Timeliness:** We quantify timeliness costs based on the average queueing delay $\tau$. We associate no timeliness costs, if $\tau$ is below or equal the target point $\tau_{tp} = 0$ sec. Therefore, we use the following cost function:

$$Q_{Timeliness}(\tau) = \begin{cases} \dfrac{\tau}{\tau_{wp}}, & \text{if } \tau > \tau_{tp} \\ 0, & \text{otherwise} \end{cases}$$

- **Provisioning:** We create the following cost function for $N_{instances}$ provisioned instances, using a the target point $P_{tp} = 1$ instance to consider, that at least 1 instance is required to let the cloud application be available:

$$Q_{Resources}(P) = \begin{cases} \dfrac{P}{P_{wp}}, & \text{if } P \geq P_{tp} \\ 0, & \text{otherwise} \end{cases}$$

The cost functions are reconstructed in each scenario by readjusting the worst point for each cost function. The worst point is obtained for each dimension by solving the overload situation by provisioning resources, adjusting the transmission rate or by accumulating messages. This enables to fairly capture each dimension, to make them comparable and to investigate the QoS characteristics of each approach. Therefore, the QoS costs should be exactly 1 if the overload situation is solely addressed by resource provisioning, transmission rate adjustments or by the initial resource configuration (baseline). Figure 8.17 illustrates the approach.

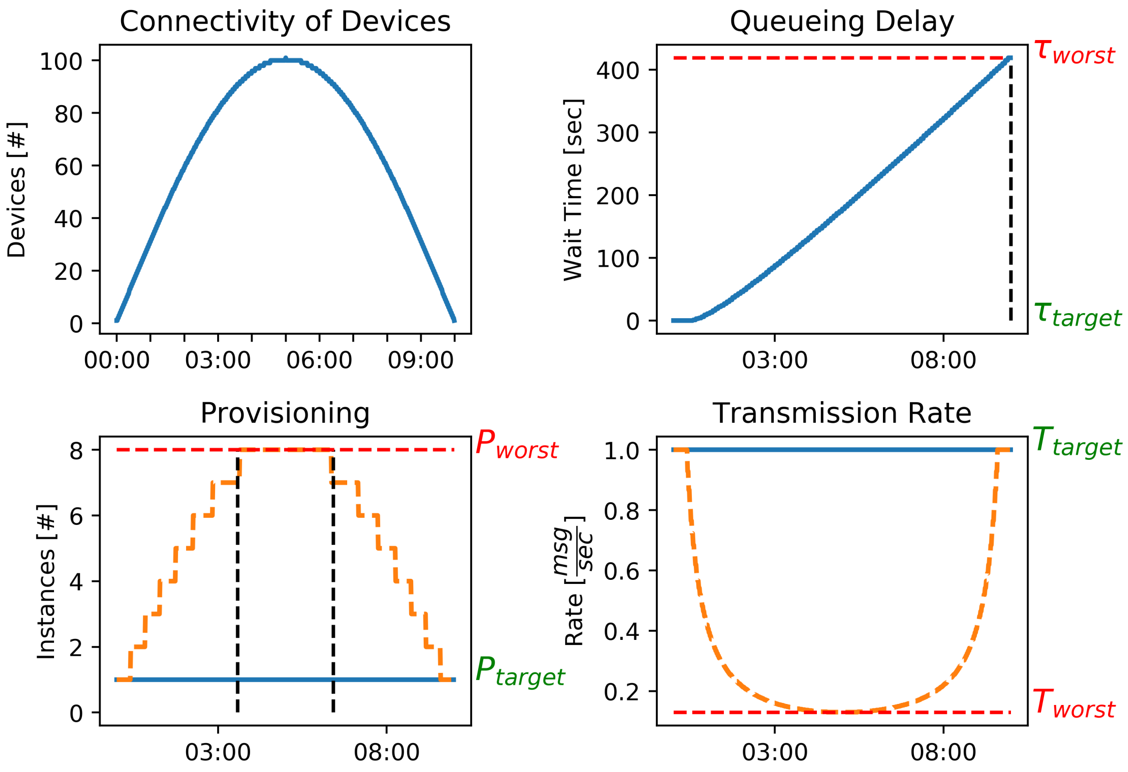


Figure 8.17.: Illustration of obtaining the worst value points. In a baseline setup, the overload situation results in a high queueing delay and peaking in $\tau_{worst}$. If the overload situation is addressed by provisioning resources, it peaks in $P_{worst}$. If the transmission rate is adjusted accordingly, it is degraded up to $T_{worst}$

.

In order to obtain the worst points for auto-scaling and flow control, we compute the optimal adjustments. Let $c$ be available cloud service capacity, which specifies the number of messages which can be processed during a time unit without inducing QoS violations in time. Then, we compute the adjustments at a time point $t$ as follows:

- **Flow-Control:** The optimal transmission rate $T(t)$ depends on the capacity of the cloud service $c$ and is computed as follows: $T(t) = min(\frac{c(t)}{N_{Dev}}, T_{tp})$.

- **Auto-Scaling:** Let $c_i$ be the capacity of a single cloud service. Then, the required instances $p(t)$ are based on and the number of connected devices $N_{Dev}$ and the transmission rate target point $T_{tp}$, such that: $p(t) = \left\lceil \frac{N_{Dev}(t) * T_{tp}}{c_i} \right\rceil$.

We obtain the average number of provisioned instances or the average transmission rate by computing it for each overload scenario. We obtain the input for the timeliness cost function by measuring the average processing delay in a static provisioning scenario.

**Connectivity and Capacity Scenario.** We configure a deterministic processing rate of $c_i = 13 \frac{\text{msg}}{\text{sec}}$ per cloud service instance with a compute-intensive message processing. In order to have time-varying behavior for the connectivity, we rely on a connectivity pattern with an increasing, plateauing and decreasing number of devices. We adjust the intensity using a multiplicator $k$ and variate $k$ in a range of $[0, 180]$ in order to create overload situations, which vary in share and intensity.

**Congestion and QoS Observer.** The congestion observer to monitors the queue length with a moving average size of 10 and a threshold of 10. The congestion observer retrieves every second an update from the infrastructure. Coupling approaches which rely on QoS costs, utilize a QoS observer with a moving average of size 10 for each input metric. The current QoS costs are recalculated once per second on the basis of the provided cost functions. Therefore, it calculates the current accuracy, timeliness and resource costs based on transmission rate, queueing delay and active instance measurements in an observation period of $T = 10$ s.

**Adaptation Protection Time.** The adaptation protection time for each strategy, including auto-scaling, is $\tau_{Adaptation} = 10$ s. This time is derived by the monitoring infrastructure latency of the message queue.

**Optimization.** For each intensity $k$ the QoS cost functions are reconstructed and normalized. The cumulative costs $Q_{Total} = Q_{Acc} + Q_{Time} + Q_{Res}$ are used for quantifying the fitness of each approach. We rely on an experience-based iteration depth of 100 and an initial population size of 40 using a DE based search.

**Configuration Space.** We optimize the parameters of the flow control approaches in $N_{flow-control}$ dimensions and of auto-scalers in $N_{auto-scaling}$ dimensions. Each coupling candidate is optimized in $N_{coupling}$ dimensions, whereas $N$ consists of $N_{flow-control}$, $N_{auto-scaling}$ and $N_{coupling-params}$ parameters, such that:

$$N_{coupling} = N_{flow-control} + N_{auto-scaling} + N_{coupling-params}$$

Note, that only the QoS rules-based coupling offers a configuration space $N_{coupling-params}$, whereas the fuzzy rules are based on fixed rules and the concurrent coupling provides no parameters. The approaches consist of the following parameter tuples $K$, which are reconfigured during the candidate search:

| Strategy | QoS Costs | | | | Measurements | | |
|---|---|---|---|---|---|---|---|
| | Total | Accuracy | Timeliness | Resources | Transmission Rate | Queueing Delay | Instances |
| TIF AIMD | 0.98 | 0.27 | 0.71 | 0.0 | 0.79 | 171.77 | 1.0 |
| Baseline | 1.0 | 0.0 | 1.0 | 0.0 | 1.0 | 226.77 | 1.0 |
| CEF | 1.02 | 0.99 | 0.03 | 0.0 | 0.2 | 7.39 | 1.0 |
| TIF-L AIMD | 1.05 | 0.97 | 0.07 | 0.0 | 0.19 | 11.9 | 1.0 |
| Reg | 1.12 | 0.0 | 0.01 | 1.12 | 1.0 | 1.4 | 9.93 |
| React (CPU) | 1.12 | 0.0 | 0.21 | 0.91 | 1.0 | 55.17 | 7.87 |

Table 8.9.: Isolated Overload Protection Approaches – Average QoS costs and measurements across all overload scenarios.

- **CEF:** Uses an overload protection multiplicator $k_{protect}$ to accelerate the congestion recovery and a recovery multiplicator $k_{recover}$ to refine the current capacity estimation: $K_{CEF} = (k_{protect}, k_{recover})$
- **TIF:** Uses an increase and a decrease parameter, which is, based on the scheme, of additive or multiplicative nature: $K_{TIF} = (k_{increase}, k_{decrease})$
- **React:** Uses an upper and lower threshold for scaling decisions based on the CPU utilization: $K_{React} = (lower, upper)$
- **Reg:** No configuration parameters.

**Simulation Accuracy.** We determine the simulation accuracy by comparing the predicted average cumulative QoS costs with the measured ones and express it as an average relative prediction error in percent $\delta Q_{total}$. Furthermore we show the average absolute error for the cost functions $Q_{res}$, $Q_{acc}$ and $Q_{time}$. We select the absolute error, since prediction errors can result in a high relative prediction error without affecting the cumulative QoS costs to a high degree. Furthermore, we show the average prediction error as percentage for the input metrics for the QoS cost functions: queueing delay $\delta\tau$, number of provisioned instances $\delta P_{Instances}$ and transmission rate $\delta T$.

**Experiment Duration.** The duration of each experiment is $t_{experiment} = 10\,\text{min}$.

### 8.6.2. Q.3 QoS Characteristics of Overload Protection Approaches

Figure 8.18 shows the average QoS cost for each strategy for intensifying overload situations and table 8.9 summarizes each strategy.

**React.** By provisioning a single instance at a time, React is challenged to (de-)provision resources in time. This is reflected by its QoS characteristics, which consists of 80 % of resource costs and 20 % of timeliness costs. React steadily increases the number of provisioned resources and is plateauing at 10.04 instances at an overload share of 500 % based on the fixed experiment duration. These limitations are reflected in a high average underprovisioning timeshare of 53.75 % with a relative deviation of 23.83 %. In low intensity overload scenarios, the overall QoS costs are reduced compared to other approaches since

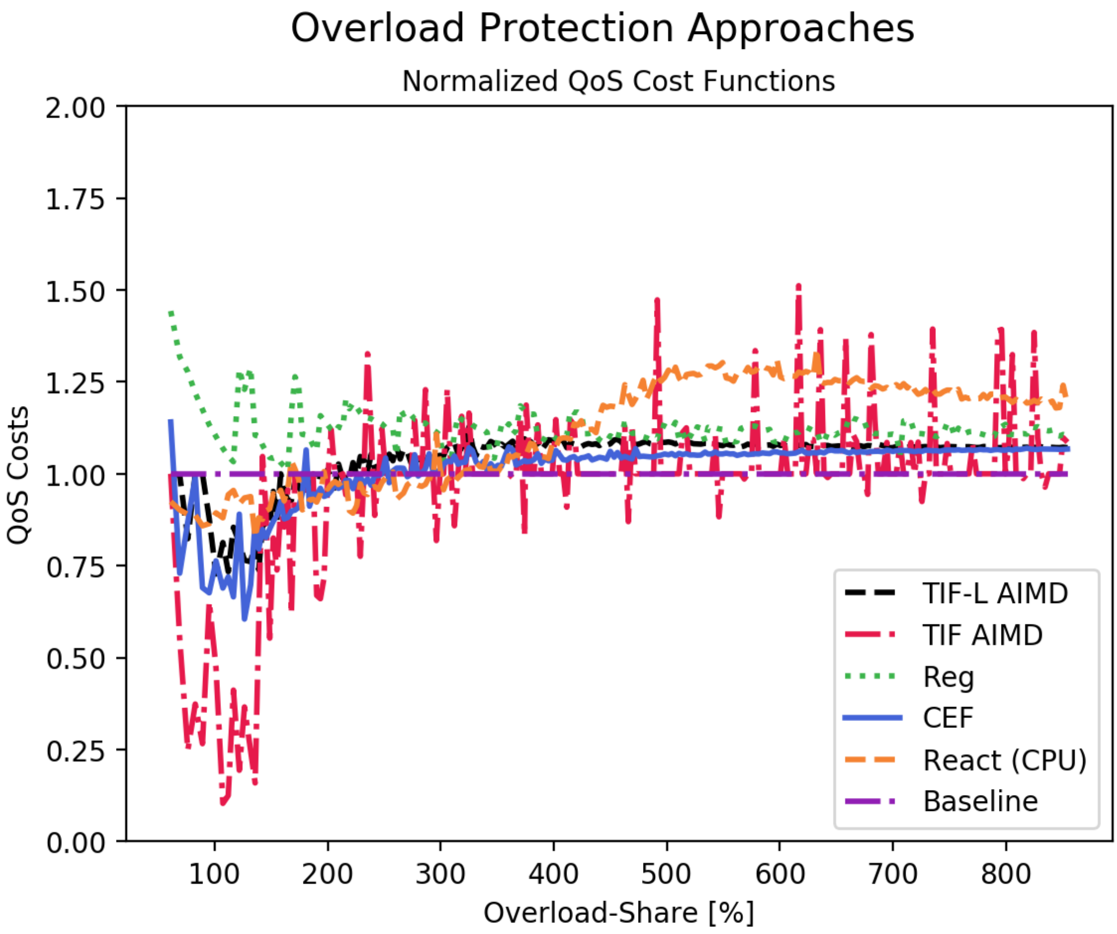


Figure 8.18.: Isolated Overload Protection Approaches – QoS Characteristics in intensifying overload scenarios.

it is slowly provision resources but resulting only in minor timeliness violations. In a high intensity overload scenarios, the QoS costs are overall decreasing. This is based on normalizing the QoS cost functions in each overload scenario, which assumes an ideal auto-scaler, which is not faced with any constraints in provisioning resources. Since the number of required instances is increasing but the provisioning of React is plateauing, the resource costs are decreasing.

**Reg.** By provisioning resources in consideration of the capacity per instance and the current load intensity, Reg is able to react to changes with a high accuracy. Furthermore it is able to react fast to an overload situation, by provisioning multiple instances at once. Therefore, the overall QoS costs are mainly resource driven with 1.13 with only minor timeliness costs of 0.01. Therefore, the scaler tends to overprovision, which is reflected by a timeshare of 62.75 % in an overprovisioned state, with a relative deviation of 18.58 %. In contrast, the average underprovisioned timeshare is 4.19 % with a very low relative deviation of 2.9 %. The regression module contributes to this by being able to predict the arrival rate changes to a high degree when faced with a steady workload decrease.

**Baseline.** By neither adjusting the transmission rate nor the provisioned resources, the baseline does only induce timeliness costs based on the induced queueing delay of the overload situation. The QoS costs are as expected on average 1, since the timeliness cost function is created by measuring the average queueing delay in each overload scenario.

**TIF AIMD.** The TIF AIMD approach holds a transmission rate state, which is shared with each smart device. As shown in the previous investigation in section 8.5, flow control approaches do induce congestions in order to probe the available capacity. The overall QoS costs are therefore expected to be composed of accuracy and timeliness violations. The optimization approach aims to minimize the cumulative QoS costs and therefore give an advantage to configurations, which only induce timeliness violations. This effectively disables the approach, but results in overall lower QoS costs. Based on the optimization depth, the data points are scattering, but overall approaching the baseline.

**TIF-L AIMD.** The TIF-L approach utilizes a load extension to hold a load or capacity model, which is fairly shared across the connected smart devices. This limits the influence of the configuration space on the adaptation behavior, since the number of connected devices influences to a strong degree the adjusted transmission rate. Overall, the approaches expenses are mainly based on accuracy degradations. However, based on behavior of the measurement provider, the congestion observer and the adaptation protection time, adaptations are not optimal, resulting in timeliness costs of 0.06, which are plateauing in overload shares over 200 % with a standard deviation of 0.01.

**CEF.** The CEF approach has similar characteristics as the TIF-L AIMD approach with a slightly improved congestion avoidance efficiency. This is reflected by the QoS costs of the accuracy and timeliness. The approach adapts overall closer to a theoretical optimal adjuster. Therefore, the accuracy costs are increased by 0.03 compared to the TIF-L AIMD approach, with decreased timeliness costs.

**Discussion.** TIF-L AIMD is nearly on-pair with CEF and both are superior to the TIF AIMD in terms of congestion-avoidance, which is reflected by low timeliness costs. However, the TIF AIMD approach offers the possibility to tune the behavior based on its parameters, resulting in a worse congestion-avoidance, which can be — within the investigation scope — beneficial in high overload scenarios. This is based on timeliness violations, which occurs in high intensity scenarios, regardless of the flow control or auto-scaling approach, increasing the cumulative QoS costs. The TIF AIMD approach allows to avoid the cumulative costs of accuracy degradation and timeliness violations by minimizing transmission rate adjustments altogether, such that costs arise only in one dimension. The resource provisioning approaches differ greatly in their behavior. Whereas Reg utilizes a capacity estimation and regression module to predict the arrival rate, React make provisioning decisions based on threshold violations. Reg is able to address the overload situation with a high adaptation accuracy, whereas React (de-)provisions a single instance at a time. In high overload scenarios, this results in lower resource costs, but higher timeliness costs compared to React.

### 8.6.3. Q.4 QoS Characteristics of Coupled Overload Protection Approaches

This section presents the result of the investigation of coupling mechanisms.

#### 8.6.3.1. Concurrent Coupling

Figure 8.19 shows the average QoS cost of each strategy for intensifying overload situations and table 8.10 summarizes each strategy.

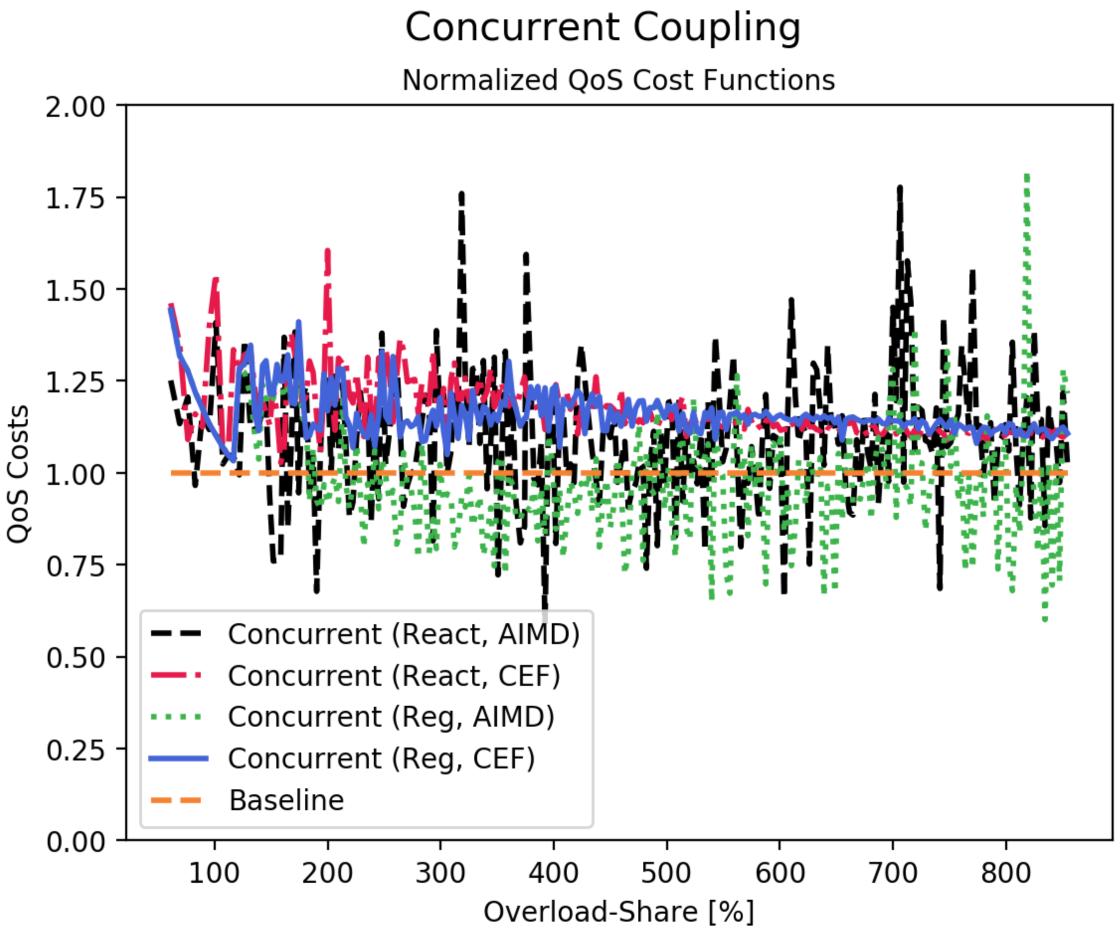


Figure 8.19.: Concurrent Coupling – QoS Characteristics in intensifying overload scenarios.

The QoS characteristics are determined by the emergent behavior of the coupled strategies. In contrast to the CEF approach, the TIF AIMD approach requires more cycles to probe the supported capacity. This allows the auto-scaling approaches to be more effective. Therefore, the QoS of coupling approaches with TIF AIMD is dominated by resource costs.

In contrast to the React, Reg provisions multiple resources at once, based on the capacity estimation and the arrival rate. This enables Reg to provision resources with a higher accuracy and with a reduced underprovisioned timeshare resulting in higher resource costs. This reduces the resource cost contribution to the overall QoS by 13 % in a CEF setup and by 34 % in a TIF setup. The reduced congestion-avoidance efficiency allows

| Strategy | QoS Costs | | | | Measurements | | |
|---|---|---|---|---|---|---|---|
| | Total | Accuracy | Timeliness | Resources | Transmission Rate | Queueing Delay | Instances |
| Concurrent (React, AIMD) | 1.1 | 0.36 | 0.04 | 0.7 | 0.69 | 10.02 | 5.68 |
| Concurrent (React, CEF) | 1.17 | 0.81 | 0.01 | 0.34 | 0.3 | 3.38 | 1.9 |
| Concurrent (Reg, AIMD) | 0.99 | 0.3 | 0.02 | 0.66 | 0.73 | 4.72 | 5.06 |
| Concurrent (Reg, CEF) | 1.16 | 0.74 | 0.02 | 0.4 | 0.35 | 4.87 | 2.24 |
| Average | 1.1 | 0.56 | 0.02 | 0.52 | 0.52 | 5.75 | 3.72 |

Table 8.10.: Concurrent Coupling – Average QoS costs and measurements across all overload scenarios.

| Strategy | QoS Costs | | | | Measurements | | |
|---|---|---|---|---|---|---|---|
| | Total | Accuracy | Timeliness | Resources | Transmission Rate | Queueing Delay | Instances |
| QoS Rules (React, TIF AIMD) | 1.03 | 0.04 | 0.91 | 0.08 | 0.97 | 214.16 | 1.16 |
| QoS Rules (React, CEF) | 1.03 | 0.05 | 0.86 | 0.13 | 0.97 | 204.16 | 1.38 |
| QoS Rules (Reg, TIF AIMD) | 1.01 | 0.04 | 0.79 | 0.17 | 0.97 | 185.05 | 1.98 |
| QoS Rules (Reg, CEF) | 0.99 | 0.07 | 0.71 | 0.22 | 0.95 | 171.38 | 2.15 |
| Average | 1.02 | 0.05 | 0.82 | 0.15 | 0.97 | 193.69 | 1.67 |

Table 8.11.: QoS-Based Coupling Rules – Average QoS costs and measurements across all overload scenarios.

to address the overload situation by resource provisioning, thus reducing the accuracy degradation to a greater extent. The CEF approach counteracts the efficiency of Reg, with the combination of both resulting in the highest cumulative QoS costs.

Overall, CEF results in disadvantages in a concurrent coupling setup based on its high congestion avoidance efficiency. The congestion avoidance is efficient enough to make provisioning unnecessary. More sophisticated auto-scaling approaches like Reg can mitigate this effect but contribute to overall high QoS costs. In combination with TIF AIMD the coupling with Reg provides a balance between resource provisioning, transmission rate adjustments and timeliness violations, resulting overall in low cumulative QoS costs. In contrast, the coupling with the CEF approach results in high resource costs and accuracy costs, providing only a minor benefit in timeliness. All approaches are able to avoid timeliness violations, with Reg coupled with TIF AIMD resulting in the best ratio of timeliness costs to resource and accuracy costs.

#### 8.6.3.2. QoS-Aware Rule-Based Coupling

Figure 8.20 shows the average QoS cost of each strategy across the overload scenarios and table 8.11 summarizes each strategy.

Since each coupled strategy is triggered by a rule set consisting of each QoS cost function and a threshold, the emerging behavior can be tuned with a fine granularity. Based on the optimization of the thresholds, the coupling approach is able to minimize and cap the QoS costs. In all overload scenarios, the activation rules of the flow control or auto-scaling tend to not trigger, resulting at QoS costs of 1 based on the timeliness degradation. Therefore, the QoS characteristics mainly comprise of timeliness violations neglecting transmission rate and provisioning adjustments. We assume, that activating the approaches result in additional timeliness degradations based on runtime dynamics, therefore it results in less QoS costs to not activate them. In contrast to a concurrent coupling, the fine-granular

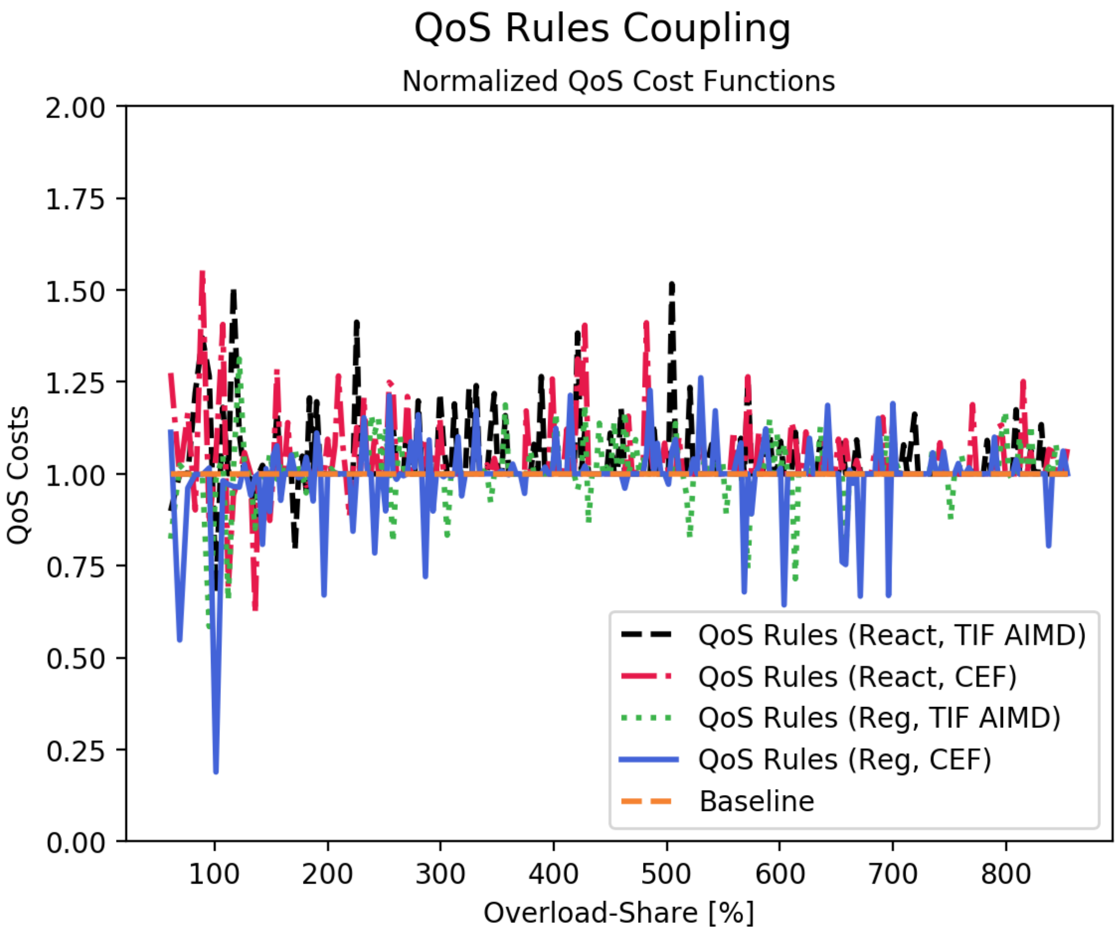


Figure 8.20.: QoS-Based Coupling Rules – QoS Characteristics in intensifying overload scenarios.

activation control allows to use timeliness as a cost sink as a mean to minimize the overall costs.

#### 8.6.3.3. Fuzzy Rules-Based Coupling

Figure 8.21 shows the average QoS cost of the strategies coupled with fuzzy rules and table 8.12 summarizes the results.

Based on the QoS cost functions, the fuzzy coupling decides when to operate flow control or auto-scaling concurrently or in isolation. The decision is based on QoS cost violations, e.g. if resource costs are bad, switch to flow control. Since the cost functions are constructed in a similar manner, it can be expected, that the costs are equally shared across resources and accuracy. However, a coupling with CEF results in high accuracy costs whereas a coupling with AIMD enables to balance the costs between the two dimensions to a greater extent. Based on the reactivity of CEF it contributes to overall higher QoS costs since it addresses overload situations with a high adaptation accuracy based on a capacity model and the current load. A coupling with the AIMD approach, which tends to

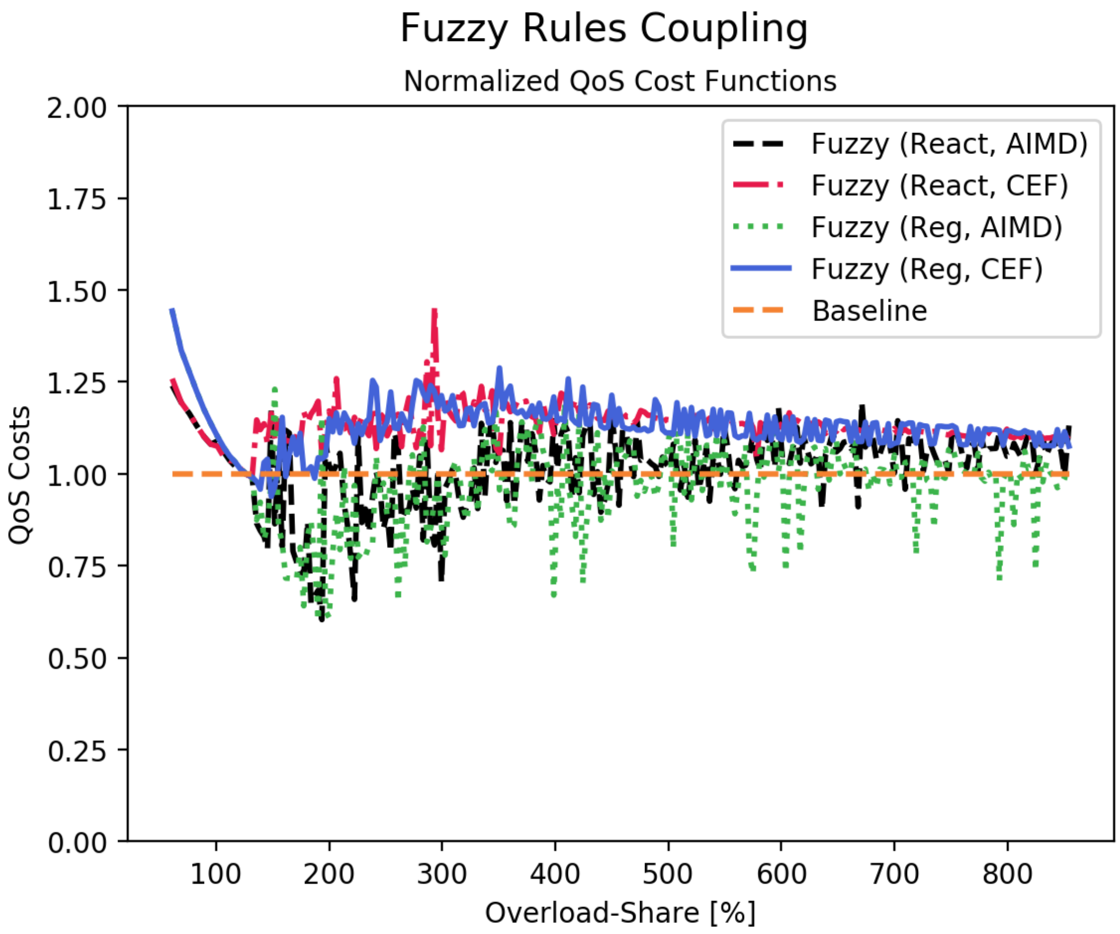


Figure 8.21.: Fuzzy Coupling Rules – QoS Characteristics in intensifying overload scenarios.

converge slowly, results in lower accuracy costs. Note, that the timeliness is increased for these couplings, since the fuzzy rules decide to utilize the AIMD approach, which in turn is too slow in addressing the overload situation, resulting in increased timeliness costs. On the other side, CEF tends to overshadow the resource provisioning, which is more obvious in the combination with React.

#### 8.6.3.4. Discussion

In this section, we have investigated the QoS characteristics of coupled approaches. The concurrent coupling is able to address overload situations with low QoS costs. The QoS characteristics differ greatly based on the combination of the approaches. Reactive and precise approaches as CEF or Reg tend to dominate the coupled approach. The coupling reflects the relation of timeliness and accuracy of adaptations of each approach. Therefore AIMD and React provides a balance, since both approaches require time to address the overload situation. The QoS-based rules coupling allows a fine-tuning of the coupling based on activation thresholds for each coupled strategy based on QoS cost functions.

| Strategy | QoS Costs | | | | Measurements | | |
|---|---|---|---|---|---|---|---|
| | Total | Accuracy | Timeliness | Resources | Transmission Rate | Queueing Delay | Instances |
| Fuzzy (React, AIMD) | 1.02 | 0.29 | 0.06 | 0.67 | 0.75 | 15.03 | 5.43 |
| Fuzzy (React, CEF) | 1.14 | 0.74 | 0.01 | 0.39 | 0.36 | 2.15 | 2.13 |
| Fuzzy (Reg, AIMD) | 0.97 | 0.43 | 0.06 | 0.48 | 0.62 | 14.92 | 3.61 |
| Fuzzy (Reg, CEF) | 1.13 | 0.82 | 0.01 | 0.3 | 0.29 | 3.12 | 1.77 |
| Average | 1.06 | 0.57 | 0.03 | 0.46 | 0.5 | 8.8 | 3.23 |

Table 8.12.: Fuzzy Coupling Rules – Average QoS costs and measurements across all overload scenarios.

| Strategy | $\delta Q_{total}$ | $\Delta Q_{acc}$ | $\Delta Q_{time}$ | $\Delta Q_{res}$ | $\delta T$ | $\delta \tau$ | $\delta Instances$ |
|---|---|---|---|---|---|---|---|
| TIF AIMD | 29.28 | 0.3 | 0.21 | 0.0 | 64.7 | 56.04 | 0.0 |
| CEF | 22.08 | 0.14 | 0.15 | 0.0 | 10.6 | 72.1 | 0.0 |
| TIF-L AIMD | 22.96 | 0.29 | 0.05 | 0.0 | 25.38 | 51.98 | 0.0 |
| RegConfig | 16.46 | 0.0 | 0.03 | 0.11 | 0.0 | 74.94 | 10.18 |
| AutoScaler-MS_CPU | 14.32 | 0.0 | 0.5 | 0.53 | 0.0 | 53.24 | 304.1 |
| Baseline | 8.68 | 0.0 | 0.1 | 0.0 | 0.0 | 8.68 | 0.0 |

Table 8.13.: Average prediction errors of the QoS costs and the input metrics as percentage difference ($\delta$) or absolute prediction errors ($\Delta$) for the flow control and auto-scaling approaches.

Therefore, the optimization is able to highly optimize the strategy to the scenario, resulting in low QoS costs. In contrast to the fuzzy and concurrent coupling, the approach is able to utilize timeliness as a cost sink based on inactivity of the coupled strategies. The fuzzy coupling utilizes a simplistic set of rules in order to decide, when to operate the coupled strategies in isolation. Since the rule set utilizes reusable rules, we expect it to be more suitable to manage different application scenarios. Overall, we recommend a concurrent coupling since both the QoS-based and the fuzzy coupling are challenging in their configuration or operation. Whereas the QoS-based coupling requires a fine-granular tuning of the weights for the activation rules for each coupled strategy, the fuzzy coupling demands an online estimation of the QoS costs, which can be challenging to provide at runtime. Additionally, in both cases the coupled approaches have to be configured on their own to achieve a high QoS conformance of the coupling. Therefore the concurrent coupling exhibits the lowest effort for a service operator and achieves a comparable high QoS conformance.

### 8.6.4. Q.5 Simulation Model Accuracy

The simulation prediction of each coupling approach is validated by conducting 4 measurements in different overload shares $overload_{share} = \{75, 150, 200, 300\}$.

**Isolated Approaches.** In the following, we validate the simulation accuracy in predicting the QoS costs for isolated approaches, e.g. auto-scaling or flow control only. The results are summarized in table 8.13.

| Strategy | $\delta Q_{total}$ | $\Delta Q_{acc}$ | $\Delta Q_{time}$ | $\Delta Q_{res}$ | $\delta T$ | $\delta \tau$ | $\delta Instances$ |
|---|---|---|---|---|---|---|---|
| Rules (Reg, TIF AIMD) | 37.75 | 0.06 | 1.06 | 0.58 | 5.33 | 98.78 | 237.0 |
| Rules (Reg, CEF) | 35.75 | 0.07 | 1.01 | 0.55 | 6.35 | 94.3 | 229.13 |
| Rules (React, TIF AIMD) | 31.0 | 0.05 | 0.78 | 0.58 | 2.63 | 71.88 | 205.02 |
| Concurrent (React, TIF AIMD) | 29.95 | 0.22 | 0.07 | 0.39 | 22.27 | 290.55 | 29.97 |
| Rules (React, CEF) | 28.45 | 0.0 | 0.56 | 0.24 | 0.17 | 50.68 | 52.08 |
| Concurrent (Reg, TIF AIMD) | 24.82 | 0.05 | 0.0 | 0.23 | 3.02 | 150.65 | 16.47 |
| FuzzyCouplingConfig (Reg, TIF AIMD) | 18.83 | 0.15 | 1.06 | 0.78 | 9.23 | 91.8 | 105.65 |
| Concurrent (React, CEF) | 16.73 | 0.42 | 0.01 | 0.27 | 37.25 | 208.35 | 38.7 |
| FuzzyCouplingConfig (React, CEF) | 12.7 | 0.56 | 1.11 | 0.61 | 39.72 | 99.8 | 76.65 |
| FuzzyCouplingConfig (Reg, CEF) | 10.2 | 0.28 | 1.16 | 0.85 | 19.58 | 99.75 | 148.13 |
| FuzzyCouplingConfig (React, TIF AIMD) | 8.75 | 0.19 | 1.1 | 0.81 | 13.15 | 98.4 | 150.0 |
| Concurrent (Reg, CEF) | 4.73 | 0.1 | 0.0 | 0.07 | 13.93 | 75.42 | 11.93 |

Table 8.14.: Average prediction errors of the QoS costs and the input metrics as percentage difference ($\delta$) or absolute prediction errors ($\Delta$) for the coupling approaches.

Overall, the simulation is able to precisely predict the behavior of the approaches resulting in a QoS cost prediction errors of less than 30.0 %. It is most accurate for the baseline scenario which has except for the varying number of devices no dynamics. In general, flow control approaches are less accurately predicted than auto-scalers, based on inaccuracies in simulating the RabbitMQ message queue of the ShapeShifter case study.

**Coupled Approaches.** In the following we validate the prediction accuracy of the coupling approaches for each of the 4 auto-scaler and flow control combinations. We validate the concurrent, the QoS-based rules and the fuzzy coupling. The results are summarized in table 8.14.

Overall, the simulation is able to predict the behavior of the approaches with a high accuracy resulting in QoS cost prediction errors of less than 40.0 %. The simulation is inaccurate in predicting the behavior of Reg coupled with a flow control approach by a set of QoS rules with a high error in predicting the provisioned instances. Since the approach relies on QoS cost functions, inaccuracies in predicting the input metrics may result in time-varying behavior of the approach. In a concurrent setup each strategy is continuously activated, thus excluding activation dynamics as a threat to the accuracy. Therefore, the concurrent approach can be predicted with a high accuracy, with an average prediction error of the cumulative QoS costs of $< 30$ %, whereas the QoS-based coupling rules exhibit a minimum error of 28.45 %.

### 8.6.5. Threats to Validity

This section discusses threats to validity of the investigation. Threats to validity related to the case study system are omitted, since they have been discussed in the previous investigation in section 8.5.6.

**External validity.** We deem accuracy, timeliness and resource costs as relevant dimensions for IoT applications which are impacted by resource provisioning and transmission rate adjustments. In order to compare the approaches to each other we have constructed normalized QoS cost functions which enable to characterize the QoS costs for each approach in intensifying overload scenarios. The characteristics of each approach are important to understand since they determine the behavior of a coupled operation to a great extent. One of the findings was, that approaches which excel in isolation can be disadvantageous in a coupled setup by dominating other approaches. We deem the findings as relevant for other IoT applications since they capture the impact and interaction of the approaches in a basic sensing cloud application.

**Construct validity.** The QoS cost functions are constructed by using a target and a worst case value. They assume a linear relation between input metric and resulting QoS costs. We derive the input for the QoS functions by monitoring the adjusted transmission rate, the number of provisioned resources and the queueing delay. In the given application scenario, we do not consider an environmental model in order to quantify the accuracy degradations by the transmission rate reduction. Therefore, we assume, that there is a linear relation between transmission rate and accuracy degradation. This is not the case for every sensing application, which may exhibit a complex and changing relation between transmission interval and accuracy. However, since the SLA of cloud applications may impose a target transmission interval and penalize degradations based on implicit accuracy assumptions we deem it as a suitable operational measure. The queueing delay is affected by overload situations and therefore a suitable proxy metric for congestion-induced delays, which affect the timeliness. The number of provisioned resources can be usually mapped to operating costs based on the contract with the cloud provider. Therefore, we deem the number of provisioned resources as suitable to be used in the QoS resource cost function. However, an influencing factor on the timeliness could be network delays and device latencies affecting the quality of adaption decisions. These factors are not considered in the current investigation, but could be included by end-to-end measurements in future ones. However, based on the experimental setup within a single cloud infrastructure, we do not think, that they affect the investigation to a greater extent.

**Reliability validity.** To the best of our knowledge, we provided all details needed to replicate the experimental setup. For this reason, we strongly expect the results to be reproducible.

### 8.6.6. Discussion

This section investigated the QoS characteristics of isolated and coupled overload protection approach and compared it to state of the art auto-scalers. We optimized each approach in each overload scenario based on normalized QoS cost function sets. Therefore, it enabled to investigate the composition of QoS costs of each approach in intensifying overload scenarios. In isolation, the resulting accuracy costs of the flow control approaches are comparable to resource costs caused by auto-scalers (Goal 3). A coupling of flow control and auto-scaling can be beneficial to the overall QoS costs since it allows to tune the adaptation behavior to a greater extent (Goal 4). Whereas the QoS characteristics in a concurrent setup are determined by emergent behavior of the coupled approaches, a rule-based approaches enable to control them to a specific degree. Rule-based approaches aim to minimize the QoS costs by deactivating the approaches altogether, since the resulting timeliness costs are less expensive than multi-dimensional costs caused by adaptations. An extension of this investigation could weight the normalized QoS cost functions in favor of a specific dimension to leverage the capabilities of the coupling approaches. We conducted controlled experiments on the Bosch IoT Cloud to validated the simulation results, which have shown a satisfying accuracy (Goal 5). Overall, the time-driven flow control approaches are able to maintain the timeliness on the expense of the accuracy. By normalizing the QoS cost functions we aimed to compare them fairly to auto-scaling. The results show, that they are similar efficient in coping with overload scenarios. Coupling the approaches can be beneficial by enabling to share timeliness degradations in overload situations on both the accuracy and resource dimension. However, in the evaluated scenario we provided linear cost functions which do not reward such behavior. Based on the satisfying QoS costs achieved by a concurrent coupling and the comparable small configuration space we recommend service operators to use this coupling mechanism.

## 8.7. QoS Contributions of time-driven Flow Control in different Application Scenarios

This section addresses Goal 6 by investigating the capabilities of (coupled) flow control approaches to maintain the QoS costs in intensifying overload scenarios for time, accuracy and resource cost driven sensing cloud applications. We represent these scenarios by weighting the QoS cost function of the specific dimension. In contrast to the investigation in section 8.6 we do not normalize the QoS cost functions in each overload scenario. Instead, we obtain a single normalized QoS cost function set for a specific overload scenario, weight it according to the application scenario and use it in each overload scenario. This aims to represent the state of the practice to a greater extent, since QoS costs functions are derived from a SLA. As in the previous investigation, we compare the approaches with auto-scaling. We obtain the results by simulation and conduct controlled experiments on the Bosch IoT Cloud to validate them.

### 8.7.1. Experimental Design

The experimental design is based on the previous one described in section 8.6. Therefore, it is refined in order to target Goal 6 of the validation. We provide a time-varying connectivity pattern and select a specific overload scenario for which we create a set of normalized cost functions. This set is used in all overload scenarios and adjusted in order to represent the QoS requirements of a specific application scenario. In contrast to the previous investigation, we aim to penalize a processing delay to a greater extent, by using the percentage of SLA violations as QoS input metric for the timeliness cost function. The application scenarios are represented by weighting the cost functions of the corresponding QoS dimensions. As in the previous investigation, we search for optimal configurations of the approaches in each overload scenario.

**Application Scenarios** We consider the the following application scenarios to be particularly relevant for sensing cloud applications:

- **Accuracy driven.** We characterize an accuracy driven application scenario as tolerable to delays and resource costs, but less to accuracy degradations. An example are environmental monitoring applications, which do not require immediate actuation.
- **Time driven.** We deem a time driven application scenario as tolerable to accuracy degradations and resource costs, but less to delays. This is relevant for smart home applications, in which the system needs to react to changes in the environment by immediate actuation.
- **Cost driven.** We consider a cost driven application scenario as tolerable to accuracy degradations and delays, but less to resource costs. E.g. applications, in which the service operator has a limited budget and wants to optimize the overall costs by degrading data qualities.
- **Mixed.** A mixed application scenario considers all dimensions as equally important. Therefore, none of the accuracy, delays and resource cost functions is weighted.

**QoS Cost Functions.** We reuse the QoS cost function creation described in section 8.6. We select an overload scenario which results in an overload share of 148 %. The constructed cost functions are the base of the optimization of each approach at any overload share.

Furthermore, we refine the timeliness cost function creation by constructing it using SLO violations. We define a SLO violation, if the processing time per message exceeds 8 s. The set point is 33 %, which means, that the timeliness cost function induced costs of 1 if 33 % of the messages are not processed in time.

All of the QoS cost functions are created using a quadratic cost function, which we realize by multiplying each cost function with itself. This rewards approaches, which are able to address the overload situation without exhausting the costs of a single quality. Therefore, a non-ideal auto-scaler which cause resource and timeliness costs is expected to perform better than accumulating messages in a static resource scenario, which burdens timeliness only. The QoS costs can rise quickly, if they exceed the set point of the cost function, e.g. if the SLO violations exceeds 66 % the timeliness costs are $\geq$ 4.

**QoS Cost Function Weighting.** We use the QoS cost set $\chi_1$ presented in chapter 4 and introduce weights $k_{acc}$, $k_{time}$ and $k_{res}$ as follows:

$$\chi_1 = k_{acc} * Q_{Accuracy} + k_{time} * Q_{Timeliness} + k_{res} * Q_{Resources}$$

Based on the concrete application scenario, the weights of each cost function are readjusted. We weight corresponding QoS dimensions with $e = 2$ in order to represent application scenarios:

- **Accuracy driven.** Let $k_{acc} = e$, $k_{time} = 1$ and $k_{res} = 1$.
- **Time driven.** Let $k_{acc} = 1$, $k_{time} = e$ and $k_{res} = 1$.

- **Cost driven.** Let $k_{acc} = 1$, $k_{time} = 1$ and $k_{res} = e$.
- **Mixed.** Let $k_{acc} = 1$, $k_{time} = 1$ and $k_{res} = 1$.

**Connectivity Pattern.** In order to have time-varying behavior we rely on a complex connectivity pattern, which is illustrated in figure 8.22. We adjust the intensity by varying a multiplicator $k$ in a range of $[1, 170]$ in order to create overload situations, which vary in share and intensity.

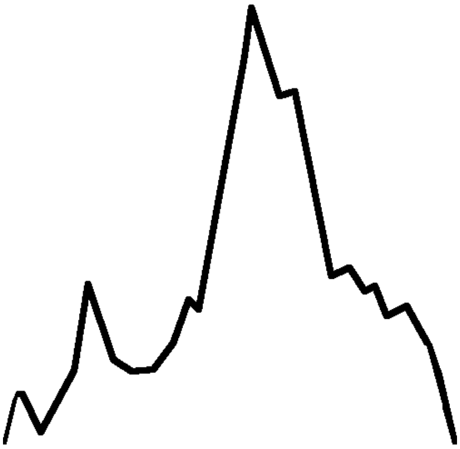


Figure 8.22.: Time-varying connectivity pattern of the investigation.

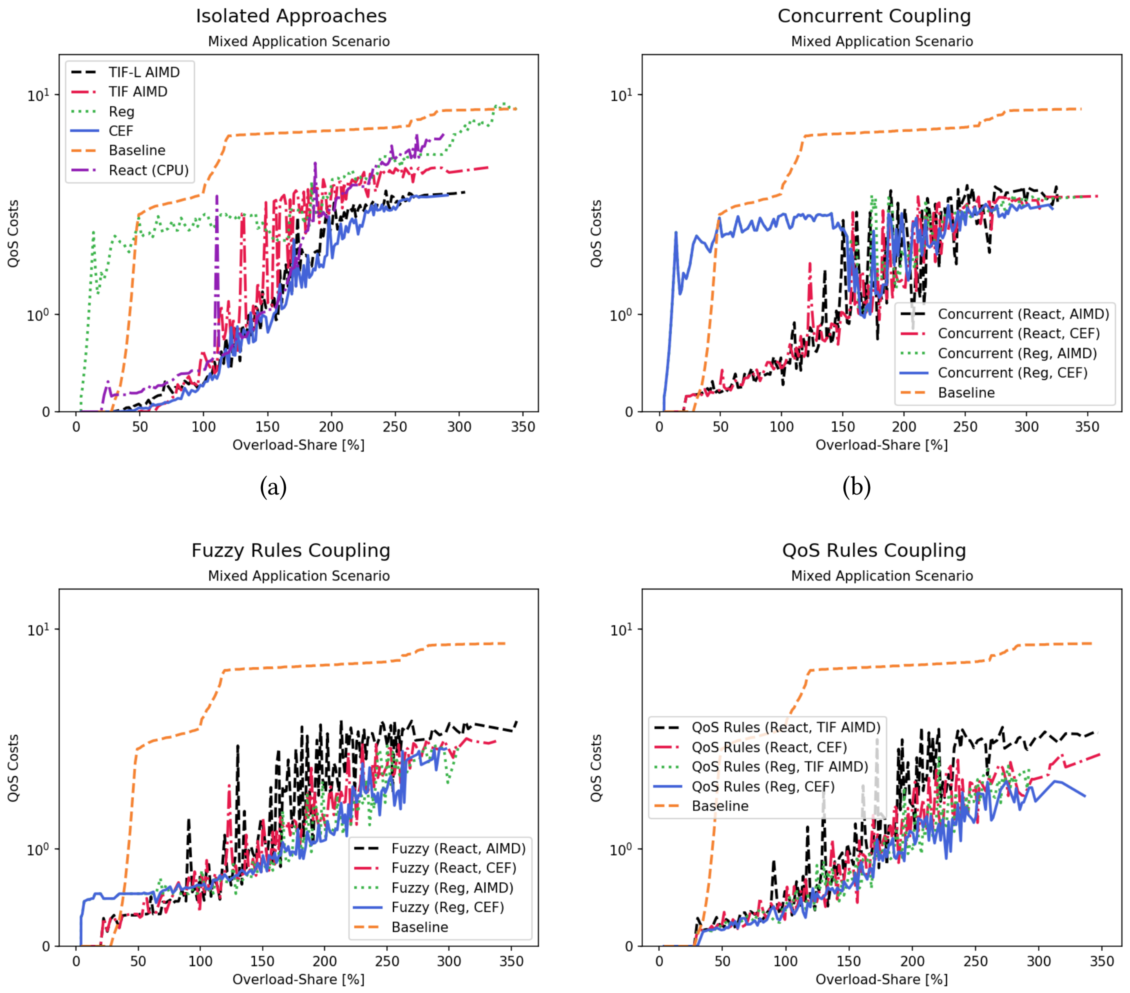
**Optimization Methodology** Based on the QoS cost functions of each application scenario, we optimize the parameters of each approach using Differential Evolution in an overload scenario with an overload share of 148 %. Therefore we obtain an optimized configuration for each approach in each application scenario for an initial overload situation. The intensity of the overload situation is then adjusted to evaluate the QoS contributions of each approach.

**Experiment Duration.** The duration of each experiment is $t_{experiment} = 10\,\text{min}$.

### 8.7.2. Q.6.1 – QoS conformance in mixed application scenarios

Figure 8.23 shows the cumulative QoS cost for each approach in a mixed application scenario. The QoS costs and measurements of the approaches are summarized in table 8.15. In the following, we discuss the results of each approach in a mixed scenario.

**Isolated.** CEF and TIF-L AIMD results in the lowest overall QoS costs within the group of isolated approaches. This is based on their high congestion-avoidance efficiency, which is able to probe the available capacity with a high precision and share it fairly across connected smart devices. Both approaches have a comparatively low share of SLO violations of 5 %. Since they adapt with a high precision, the accuracy costs exceeds 1 only slightly. As in the



(a)

(b)

(c)

(d)

Figure 8.23.: Mixed Application Scenario. Cumulative QoS Costs of each approach in intensifying overload scenarios.

previous investigations, the CEF approach is slightly better, by an overall cost reduction of 7.6 %.

By being sensitive to connectivity variations, the TIF AIMD approach is prone to induce long lasting congestions. This is reflected by high timeliness costs based on average SLO violations of 19 %. Therefore, the inferior congestion-avoidance characteristics result in overall higher costs.

The Reg approach utilizes a similar mechanism as CEF in order to estimate the capacity and adjust the provisioning in respect to the load intensity. Overall, it reacts fast and efficient, avoiding SLO violations altogether, which are at 0 %. However, as concluded in the previous investigation, it tends to overprovision, which is strongly penalized by the cost functions. Whereas it is more efficient than the flow control approaches in avoiding congestions, it is less accurate and therefore overall more expensive. Especially in low overload scenarios, it tends to eagerly provision resources, reflected by high initial costs.

| Approach | QoS Costs | | | | Measurements | | |
|---|---|---|---|---|---|---|---|
| | Total | Accuracy | Timeliness | Resources | Transmission Rate [$\frac{msg}{sec}$] | SLO Violations [%] | Instances [#] |
| CEF | 1.58 | 1.53 | 0.05 | 0.0 | 0.57 | 5.74 | 1.0 |
| TIF-L AIMD | 1.69 | 1.62 | 0.06 | 0.0 | 0.56 | 5.87 | 1.0 |
| TIF AIMD | 3.07 | 2.16 | 0.91 | 0.0 | 0.51 | 12.71 | 1.0 |
| Reg | 3.46 | 0.0 | 0.0 | 3.46 | 1.0 | 0.0 | 4.68 |
| React (CPU) | 4.42 | 0.0 | 1.31 | 3.11 | 1.0 | 10.94 | 4.07 |
| Baseline | 5.46 | 0.0 | 5.46 | 0.0 | 1.0 | 72.3 | 1.0 |
| Average | 3.28 | 0.89 | 1.3 | 1.1 | 0.77 | 17.83 | 2.13 |
| | | | | | | | |
| Concurrent (React, CEF) | 1.94 | 0.87 | 0.06 | 1.01 | 0.7 | 1.37 | 2.15 |
| Concurrent (Reg, CEF) | 1.99 | 0.65 | 0.0 | 1.34 | 0.77 | 0.47 | 2.95 |
| Concurrent (React, AIMD) | 2.31 | 0.44 | 0.58 | 1.29 | 0.81 | 5.01 | 2.6 |
| Concurrent (Reg, AIMD) | 2.25 | 0.34 | 0.13 | 1.78 | 0.85 | 1.96 | 3.42 |
| Average | 2.12 | 0.57 | 0.2 | 1.35 | 0.78 | 2.2 | 2.78 |
| | | | | | | | |
| Fuzzy (Reg, AIMD) | 1.34 | 0.28 | 0.12 | 0.94 | 0.84 | 1.72 | 2.48 |
| Fuzzy (Reg, CEF) | 1.36 | 0.42 | 0.0 | 0.94 | 0.79 | 0.49 | 2.49 |
| Fuzzy (React, CEF) | 1.56 | 0.77 | 0.09 | 0.7 | 0.72 | 1.42 | 2.09 |
| Fuzzy (React, AIMD) | 1.94 | 0.4 | 0.36 | 1.17 | 0.82 | 4.03 | 2.62 |
| Average | 1.55 | 0.47 | 0.14 | 0.94 | 0.79 | 1.92 | 2.42 |
| | | | | | | | |
| QoS Rules (Reg, CEF) | 1.15 | 0.52 | 0.07 | 0.55 | 0.75 | 1.57 | 1.86 |
| QoS Rules (Reg, AIMD) | 1.27 | 0.16 | 0.17 | 0.94 | 0.88 | 2.07 | 2.38 |
| QoS Rules (React, CEF) | 1.61 | 0.68 | 0.15 | 0.79 | 0.73 | 2.03 | 2.03 |
| QoS Rules (React, AIMD) | 2.14 | 0.42 | 0.59 | 1.14 | 0.8 | 4.54 | 2.54 |
| Average | 1.54 | 0.45 | 0.24 | 0.85 | 0.79 | 2.55 | 2.2 |

Table 8.15.: Mixed Application Scenario – Average QoS costs and measurements across all overload scenarios.

This is based on the capacity estimation, which is only refined in overload situations, and relying on a guess based on the current arrival rate. Based on the low load intensity, the guess does not reflect the capacity of the service, thus resulting in overprovisioning.

The React approach degrades the QoS by being too slow and inaccurate in provisioning resources. This can be beneficial in low intensity overload situations, since the overall costs are shared across resources and timeliness, resulting in overall low cumulative QoS costs. However, the characteristics have a negative impact in high intensity overload situations, which result in a high amount of SLO violations. This is reflected by in average 27.11 % SLO violations.

Based on the quadratic cost functions, the baseline is heavily penalized by inducing SLO violations up to 100 % at an overload share of roughly 300 %. Therefore, it results in the highest overall costs of on average 5.46.

**Concurrent Coupling.** The concurrent coupling results on average in an improved QoS conformance compared to isolated approaches. As investigated in section 8.6, both, CEF and Reg are reactive and tend to overshadow the coupled approaches. In low overload scenarios up to 180 %, a coupling results in slightly higher costs compared to isolated approaches. Especially Reg coupled with CEF or coupled with TIF AIMD results in degraded costs, since Reg is more reactive than the flow control approaches by not relying on a congestion observer to trigger resource provisioning. Therefore, the flow control approaches do not contribute to congestion avoidance, which results in a nearly equivalent behavior to an isolated Reg approach. However, this changes in more intense overload situations. Then, the combination of CEF and Reg results in the lowest cumulative QoS costs, with a fair share of resource provisioning and transmission rate adjustments. Coupling React with the flow control approaches results in higher SLO violations but lower resource costs. On average, there is a clear improvement in the QoS conformance of 166 % compared to the isolated approaches. However, in less severe overload situations, the coupling may degrade the QoS conformance.

**Fuzzy Rules Coupling.** The fuzzy rules approach switches to an isolated operation of flow control or auto-scaling if the accuracy or resource costs have exceed a certain threshold. This is beneficial for a coupling of Reg with CEF or TIF AIMD, since it does contribute to worse QoS costs in low intensity overload scenarios, as observed in the concurrent coupling. By deciding to switch to flow control only, the costs are greatly reduced. Overall, it results in a clear improvement compared to the concurrent approach, by avoiding a dominant resource provisioning.

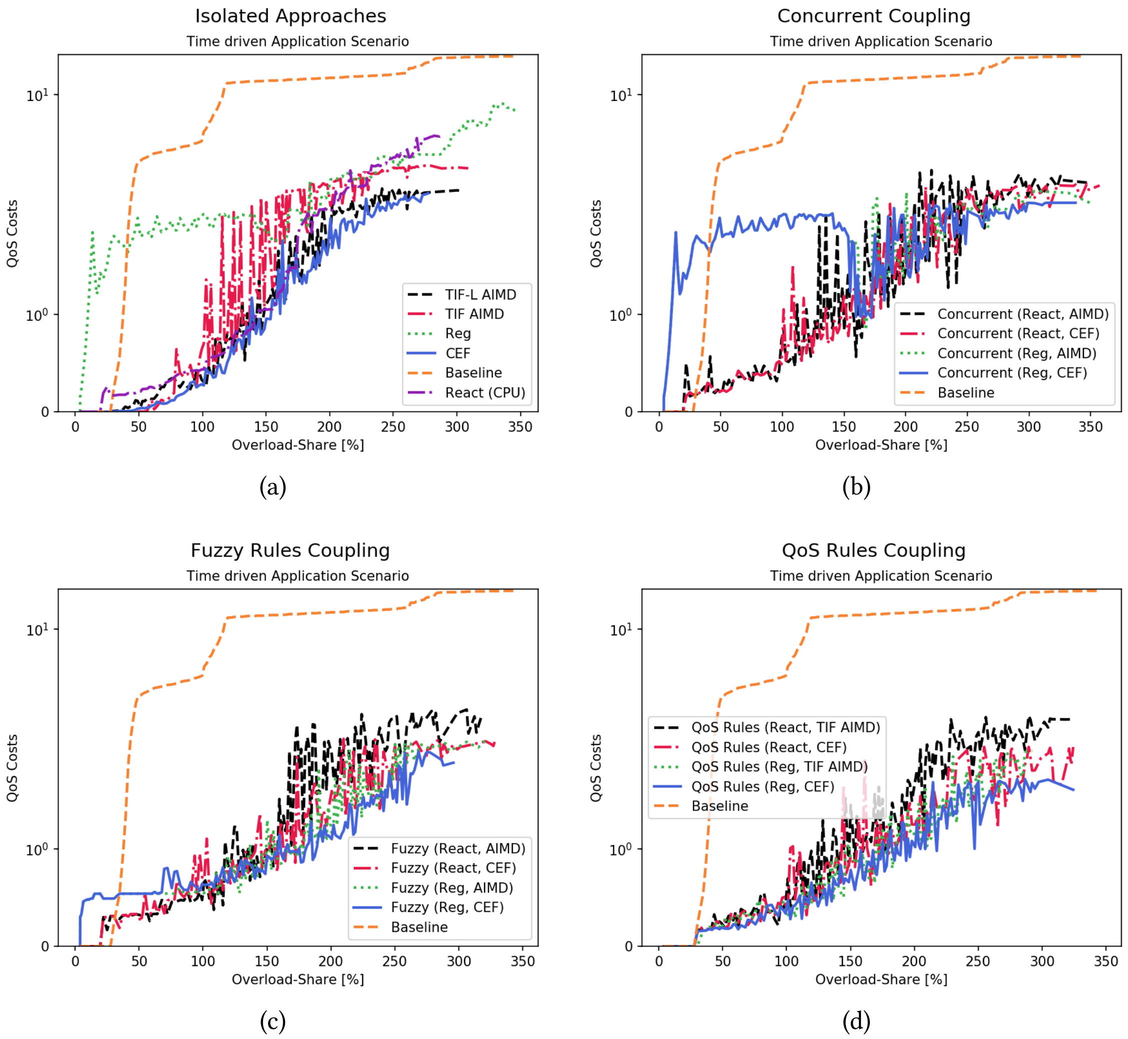
**QoS Rules Coupling.** By being optimized in each overload scenario, the fine-granular rule set of the QoS coupling aims to leverage the capabilities of an activation control coupling. Therefore, it is expected to result in the lowest QoS costs, which is the case in the experiment. Overall, it achieves the lowest average QoS costs with 1.54. The optimization results in tuned thresholds for the given scenario, which may decide to initially operate only one or none approach. This is shown in low overload intensity scenarios, since it deactivates the resource provisioning of Reg altogether, avoiding the high initial spike. The coupling of Reg and CEF results in a harmonizing behavior, in which the QoS costs are fairly shared. Overall, the activation control leverages the potential of the coupled approaches in isolation and in combination.

**Discussion.** In the mixed scenario, timeliness, accuracy and provisioning costs are equally weighted. Therefore, each approach is optimized to share the QoS costs across multiple dimensions. This is leveraged by the quadratic cost functions, which penalize approaches heavily, if they are unable to utilize multiple cost dimensions. For this reason, isolated approaches are challenged in maintaining a high QoS conformance. This is reflected by the average QoS costs of 4.33, which exceeds the QoS costs of coupled approaches by 166 % to 195 %.

### 8.7.3. Q.6.2 – QoS conformance in time driven application scenarios

Figure 8.24 shows the cumulative QoS cost of the isolated and coupled overload protection approaches in a time driven application scenario. The QoS costs and measurements of the approaches are summarized in table 8.16. In the following, we discuss the results of each approach.

**Isolated.** Approaches with a high congestion avoidance efficiency tend to avoid long lasting overload situations. Therefore, CEF, TIF-L AIMD and Reg result in the lowest overall QoS costs. Since Reg tends to overprovision in the low overload scenario, its QoS costs are degraded by inducing high resource costs. The inferior congestion avoidance



(a)

(b)

(c)

(d)

Figure 8.24.: Time driven Application Scenario. Cumulative QoS Costs of each approach in intensifying overload scenarios.

characteristics of the TIF AIMD approach result in relatively high QoS costs by inducing long lasting congestions, which are reflected by SLO violations of 14.97 %. Especially in higher overload scenarios, the React approach is unable to cope with the overload situation in time, resulting in average SLO violations of 21.06 %. As expected, the baseline approach exhibits the lowest QoS conformance based on the high amount of SLO violations. Overall, there is an average QoS degradation of 17.0 % compared to the mixed setup.

**Concurrent Coupling.** A concurrent coupling inherits the timeliness properties of isolated approaches. If one of the coupled approaches is able to avoid timeliness violations to a great extent, the overall QoS costs are not affected compared to a mixed application scenario. However, if both approaches are less efficient in avoiding congestions, e.g. TIF AIMD and React, the timeliness violations contribute to 13.7 % higher QoS costs.

| Approach | QoS Costs | | | | Measurements | | |
|---|---|---|---|---|---|---|---|
| | Total | Accuracy | Timeliness | Resources | Transmission Rate [$\frac{msg}{sec}$] | SLO Violations [%] | Instances [#] |
| CEF | 1.61 | 1.54 | 0.07 | 0.0 | 0.57 | 4.64 | 1.0 |
| TIF-L AIMD | 1.76 | 1.7 | 0.06 | 0.0 | 0.54 | 3.82 | 1.0 |
| Reg | 3.48 | 0.0 | 0.0 | 3.48 | 1.0 | 0.0 | 4.69 |
| TIF AIMD | 3.74 | 2.48 | 1.26 | 0.0 | 0.46 | 14.97 | 1.0 |
| React (CPU) | 5.47 | 0.0 | 1.96 | 3.5 | 1.0 | 21.06 | 4.3 |
| Baseline | 10.92 | 0.0 | 10.92 | 0.0 | 1.0 | 72.3 | 1.0 |
| Average | 4.49 | 0.95 | 2.37 | 1.17 | 0.76 | 20.61 | 2.17 |
| | | | | | | | |
| Concurrent (Reg, CEF) | 2.02 | 0.65 | 0.0 | 1.36 | 0.77 | 0.42 | 2.98 |
| Concurrent (React, CEF) | 2.23 | 0.86 | 0.1 | 1.26 | 0.71 | 1.31 | 2.29 |
| Concurrent (Reg, AIMD) | 2.39 | 0.26 | 0.16 | 1.97 | 0.87 | 1.28 | 3.59 |
| Concurrent (React, AIMD) | 3.01 | 0.48 | 1.04 | 1.49 | 0.81 | 4.51 | 2.76 |
| Average | 2.41 | 0.57 | 0.33 | 1.52 | 0.79 | 1.88 | 2.9 |
| | | | | | | | |
| Fuzzy (Reg, CEF) | 1.34 | 0.38 | 0.01 | 0.95 | 0.8 | 0.53 | 2.5 |
| Fuzzy (Reg, AIMD) | 1.53 | 0.36 | 0.18 | 0.99 | 0.82 | 1.47 | 2.54 |
| Fuzzy (React, CEF) | 1.63 | 0.77 | 0.16 | 0.7 | 0.72 | 1.38 | 2.09 |
| Fuzzy (React, AIMD) | 2.27 | 0.54 | 0.53 | 1.2 | 0.79 | 2.91 | 2.62 |
| Average | 1.69 | 0.51 | 0.22 | 0.96 | 0.78 | 1.57 | 2.44 |
| | | | | | | | |
| QoS Rules (Reg, CEF) | 1.2 | 0.52 | 0.11 | 0.57 | 0.76 | 1.43 | 1.91 |
| QoS Rules (Reg, AIMD) | 1.42 | 0.21 | 0.24 | 0.97 | 0.87 | 1.76 | 2.41 |
| QoS Rules (React, CEF) | 1.71 | 0.81 | 0.29 | 0.62 | 0.71 | 1.97 | 1.87 |
| QoS Rules (React, AIMD) | 2.47 | 0.42 | 0.88 | 1.16 | 0.8 | 3.58 | 2.57 |
| Average | 1.7 | 0.49 | 0.38 | 0.83 | 0.78 | 2.18 | 2.19 |

Table 8.16.: Time driven Application Scenario – Average QoS costs and measurements across all overload scenarios.

**Fuzzy Rules Coupling.** The fuzzy rule set is independent to timeliness costs. Therefore, the activation mix of resource provisioning and flow control is not affected by the change in the application scenario. However, the resulting QoS costs are affected, since timeliness costs are increased. They are nearly 0 for a coupling of CEF and Reg based on their congestion avoidance efficiency. The remaning coupling candidates do induce timeliness costs, which result in a degradation of the overall cumulative QoS costs compared to the mixed application scenario by 10.9 %.

**QoS Rules Coupling.** The QoS cost characteristics in accuracy and resources remain nearly unaffected in comparison to the mixed application scenario. However, the rule set is more eager to activate approaches to avoid congestion induced SLO violations which is reflected by slightly degraded timeliness costs and slightly increased resource and accuracy costs for all approaches. However, no approach is able to avoid timeliness violations altogether resulting in overall degraded QoS costs of 10 %.

**Discussion.** In a time driven scenario, timeliness costs are heavily penalized. This rewards approaches which avoid SLO violations. The QoS costs are strongly increased for isolated approaches which are not efficient at avoiding congestions, e.g. React or TIF AIMD. A coupling based on fuzzy rules or a concurrent operation is not affected in its activation behavior based on the time driven scenario. The QoS rules are able to configure activation rules, which activate each approach more often in order to reduce timeliness violations. All approaches have resulted in higher QoS costs since each is not able to address the overload situations without inducing SLO violations.

### 8.7.4. Q.6.3 – QoS conformance in accuracy driven application scenarios

Figure 8.25 shows the cumulative QoS cost of each approach in an accuracy driven application scenario. The QoS costs and measurements are summarized in table 8.17. In the following, we discuss the results of each approach.

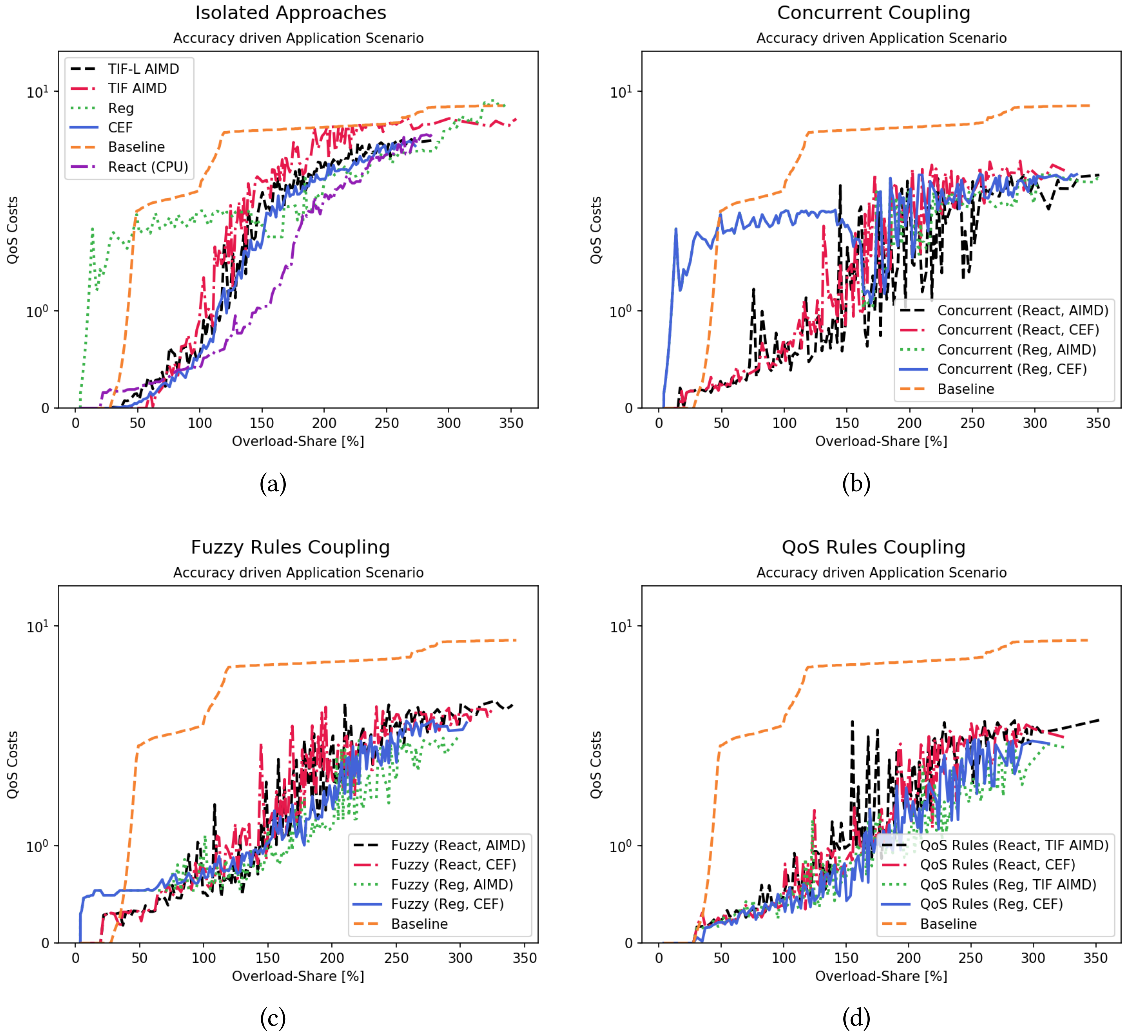


Figure 8.25.: Accuracy driven Application Scenario. Cumulative QoS Costs of each approach in intensifying overload scenarios.

**Isolated.** The weight on accuracy results in a major degradation of the QoS conformance of flow control approaches up to 100 %. Since auto-scaling does not induce accuracy costs it has a QoS cost advantage in this application scenario. However, this advantage is overshadowed for Reg by overprovisioning in low intensity overload scenarios and for React by timeliness violations in high overload scenarios. Therefore, the flow control approaches are on average only slightly below auto-scaling approaches. Notably, the TIF AIMD approach is able to reduce accuracy degradations, since the optimized parameters

| Approach | QoS Costs | | | | Measurements | | |
|---|---|---|---|---|---|---|---|
| | Total | Accuracy | Timeliness | Resources | Transmission Rate [$\frac{msg}{sec}$] | SLO Violations [%] | Instances [#] |
| CEF | 3.04 | 2.96 | 0.09 | 0.0 | 0.58 | 7.45 | 1.0 |
| TIF-L AIMD | 3.26 | 3.1 | 0.15 | 0.0 | 0.57 | 9.81 | 1.0 |
| React (CPU) | 3.42 | 0.0 | 0.29 | 3.13 | 1.0 | 10.4 | 4.08 |
| Reg | 3.46 | 0.0 | 0.0 | 3.46 | 1.0 | 0.0 | 4.68 |
| TIF AIMD | 4.64 | 1.38 | 3.27 | 0.0 | 0.78 | 44.83 | 1.0 |
| Baseline | 5.45 | 0.0 | 5.45 | 0.0 | 1.0 | 72.8 | 1.0 |
| Average | 4.03 | 1.24 | 1.69 | 1.1 | 0.82 | 24.13 | 2.13 |
| | | | | | | | |
| Concurrent (Reg, CEF) | 2.54 | 0.69 | 0.0 | 1.86 | 0.85 | 0.28 | 3.48 |
| Concurrent (Reg, AIMD) | 2.62 | 0.39 | 0.07 | 2.15 | 0.9 | 0.94 | 3.72 |
| Concurrent (React, AIMD) | 2.63 | 0.73 | 0.53 | 1.37 | 0.83 | 4.76 | 2.73 |
| Concurrent (React, CEF) | 3.09 | 1.41 | 0.06 | 1.62 | 0.74 | 1.21 | 2.55 |
| Average | 2.72 | 0.8 | 0.16 | 1.75 | 0.83 | 1.8 | 3.12 |
| | | | | | | | |
| Fuzzy (Reg, AIMD) | 1.48 | 0.28 | 0.12 | 1.08 | 0.89 | 1.65 | 2.64 |
| Fuzzy (Reg, CEF) | 1.76 | 0.64 | 0.0 | 1.12 | 0.82 | 0.39 | 2.7 |
| Fuzzy (React, AIMD) | 2.33 | 0.45 | 0.36 | 1.52 | 0.86 | 4.16 | 2.96 |
| Fuzzy (React, CEF) | 2.26 | 1.07 | 0.08 | 1.11 | 0.77 | 1.23 | 2.52 |
| Average | 1.96 | 0.61 | 0.14 | 1.21 | 0.84 | 1.86 | 2.7 |
| | | | | | | | |
| QoS Rules (Reg, AIMD) | 1.44 | 0.28 | 0.16 | 1.0 | 0.89 | 2.05 | 2.44 |
| QoS Rules (Reg, CEF) | 1.57 | 0.82 | 0.07 | 0.68 | 0.79 | 1.53 | 2.05 |
| QoS Rules (React, CEF) | 2.29 | 1.22 | 0.17 | 0.9 | 0.76 | 2.13 | 2.18 |
| QoS Rules (React, AIMD) | 2.61 | 0.66 | 0.54 | 1.41 | 0.84 | 4.19 | 2.77 |
| Average | 1.98 | 0.74 | 0.24 | 0.99 | 0.82 | 2.47 | 2.36 |

Table 8.17.: Accuracy driven Application Scenario – Average QoS costs and measurements across all overload scenarios.

do affect its congestion avoidance efficiency to a great extent, as discussed in section 8.6.2. However, this is achieved on the expense of timeliness, which result in overall increased QoS costs.

**Concurrent Coupling.** Whereas a concurrent coupling does not control the activation of approaches, the emergent behavior is tuned by the initial optimization of the configuration of the coupled approaches for the time driven application scenario. Therefore, the overall transmission rate adjustments are decreased in comparison to the mixed application scenario by 6 %, whereas the instance provisioning is increased by 23 %. The efficiency of Reg enables to maximize this effect since its resource provisioning triggers the flow control approaches to a lesser degree resulting in low cumulative QoS costs.

**Fuzzy Rules Coupling.** The fuzzy rules favor auto-scaling when faced with high accuracy costs. This is reflected by a reduced transmission rate adjustment compared to the mixed application scenario of in average 6.3 %. Based on the less efficient congestion avoidance property of TIF AIMD it acts favorable on the QoS costs since the transmission rate adjustments are less intense. Overall, the combination of Reg and CEF remains a strong coupling candidate which is excelled by Reg and AIMD since both profit from the efficient resource provisioning of Reg.

**QoS Rules Coupling.** Whereas it could be expected, that the QoS rules tend to not activate flow control approaches, they still do. This is based on the results of auto-scaling in isolation and the baseline, which contributes in this experiment to higher QoS costs

| Approach | QoS Costs | | | | Measurements | | |
|---|---|---|---|---|---|---|---|
| | Total | Accuracy | Timeliness | Resources | Transmission Rate [$\frac{msg}{sec}$] | SLO Violations [%] | Instances [#] |
| CEF | 1.57 | 1.52 | 0.05 | 0.0 | 0.57 | 1.83 | 1.0 |
| TIF-L AIMD | 1.69 | 1.63 | 0.06 | 0.0 | 0.56 | 1.71 | 1.0 |
| TIF AIMD | 3.08 | 2.32 | 0.77 | 0.0 | 0.48 | 10.45 | 1.0 |
| Baseline | 5.48 | 0.0 | 5.48 | 0.0 | 1.0 | 103.53 | 1.0 |
| Reg | 6.97 | 0.0 | 0.0 | 6.97 | 1.0 | 0.14 | 4.69 |
| React (CPU) | 7.17 | 0.0 | 1.76 | 5.41 | 1.0 | 15.42 | 3.83 |
| Average | 4.33 | 0.91 | 1.35 | 2.06 | 0.77 | 22.18 | 2.09 |
| | | | | | | | |
| Concurrent (React, CEF) | 2.6 | 1.06 | 0.08 | 1.45 | 0.66 | 1.63 | 1.84 |
| Concurrent (React, AIMD) | 3.25 | 0.69 | 0.76 | 1.8 | 0.75 | 6.34 | 2.26 |
| Concurrent (Reg, AIMD) | 3.81 | 0.59 | 0.26 | 2.95 | 0.78 | 3.07 | 3.12 |
| Concurrent (Reg, CEF) | 3.17 | 1.05 | 0.01 | 2.12 | 0.69 | 0.7 | 2.57 |
| Average | 3.21 | 0.85 | 0.28 | 2.08 | 0.72 | 2.93 | 2.45 |
| | | | | | | | |
| Fuzzy (Reg, CEF) | 2.13 | 0.71 | 0.0 | 1.42 | 0.72 | 0.58 | 2.18 |
| Fuzzy (React, CEF) | 2.18 | 0.94 | 0.09 | 1.15 | 0.68 | 1.52 | 1.87 |
| Fuzzy (Reg, AIMD) | 2.23 | 0.43 | 0.19 | 1.61 | 0.8 | 2.42 | 2.31 |
| Fuzzy (React, AIMD) | 2.99 | 0.7 | 0.51 | 1.78 | 0.76 | 5.49 | 2.32 |
| Average | 2.38 | 0.69 | 0.2 | 1.49 | 0.74 | 2.5 | 2.17 |
| | | | | | | | |
| QoS Rules (Reg, CEF) | 1.66 | 0.77 | 0.1 | 0.78 | 0.7 | 1.84 | 1.6 |
| QoS Rules (React, CEF) | 1.97 | 1.07 | 0.18 | 0.72 | 0.65 | 2.3 | 1.5 |
| QoS Rules (Reg, AIMD) | 2.14 | 0.26 | 0.22 | 1.65 | 0.84 | 2.3 | 2.23 |
| QoS Rules (React, AIMD) | 3.41 | 0.51 | 0.73 | 2.17 | 0.78 | 5.17 | 2.35 |
| Average | 2.29 | 0.65 | 0.31 | 1.33 | 0.74 | 2.9 | 1.92 |

Table 8.18.: Cost driven Application Scenario – Average QoS costs and measurements across all overload scenarios.

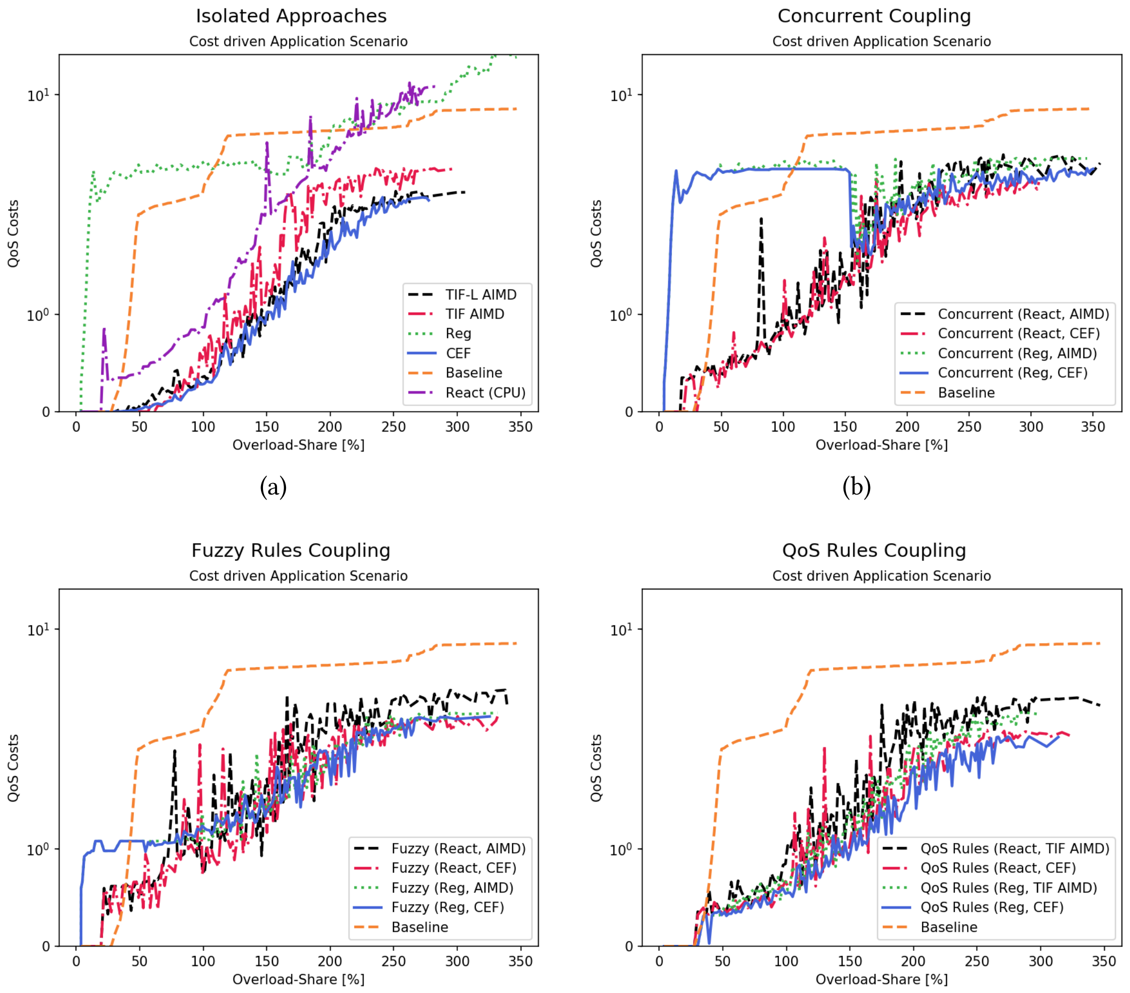
than combining it with flow control. Therefore, the coupling only slightly reduces the transmission rate adjustments on the expense of an increased resource provisioning. Overall, the weighted accuracy costs and the increased provisioning costs result in a QoS conformance degradation of in average 23 % compared to a mixed application scenario.

**Discussion.** In an accuracy driven application scenario the adjustments of time-driven flow control approaches are heavily penalized. Therefore, they induce high QoS costs if operated on their own. Coupling flow control approaches with auto-scaling has shown to be beneficial on the overall QoS.

### 8.7.5. Q.6.4 – QoS conformance in cost driven application scenarios

Figure 8.26 shows the cumulative QoS cost for each approach in a cost driven application scenario. The QoS costs and measurements of the approaches are summarized in table 8.18.

**Isolated.** Since the weighted resource cost function does only affect the auto-scaling approaches, the degradation of the QoS conformance is especially strong for Reg and React. Since Reg offers no parameters space, the resulting QoS costs are nearly doubled, whereas React is able to reduce the resource provisioning by 6 % on the expense of timeliness. Nevertheless, React results in the highest QoS costs of 7.17. Both approaches exceeds the baseline by at least 10 %.



(a)

(b)

(c)

(d)

Figure 8.26.: Cost-driven Application Scenario. Cumulative QoS Costs of each approach in intensifying overload scenarios.

**Concurrent Coupling.** In contrast to a mixed application scenario, a coupling with Reg results in worse QoS costs than a coupling with React. This is based on its reactivity and the lack of a parameter space, such that it can not be tuned for this scenario. Since CEF is more efficient in congestion avoidance than TIF AIMD, it achieves the lowest QoS costs by enabling to sink the QoS costs on accuracy.

**Fuzzy Rules Coupling.** The fuzzy rules enables to switch to flow control, if the resource costs have exceeded a certain threshold. CEF is a more suitable fit, by being able to degrade the accuracy and maintain low timeliness violations. This lighten the burden on the auto-scaling approaches. Notably, a coupling with Reg performs slightly better than a coupling with React. This may be based on the property of React to deprovision only one microservice as a time resulting in high resource costs.

**QoS Rules Coupling.** In comparison to the mixed application scenario, the provisioned instances are reduced by 15 %, which allow to avoid a strong QoS degradation based on resource costs. However, they are still considerably increased by 56 %. Nevertheless, the QoS coupling results in the lowest average QoS costs. Similar to the fuzzy rules coupling the QoS rules coupling achieves the best results with Reg coupled with CEF followed by React coupled with CEF.

**Discussion.** In a cost driven application scenario auto-scaling approaches are challenged since resource provisioning is heavily penalized. Therefore auto-scaling results in high QoS costs. They can be mitigated by relying on coupling approaches at which a QoS based rule set performs best.

### 8.7.6. Q.6.5 – Simulation Model Accuracy

The simulation prediction of each coupling approach is validated by conducting 5 measurements in different overload shares $overload_{share} = \{75, 150, 200, 300\}$ and each application scenario $scenario = \{mixed, timedriven, costdriven, accuracydriven\}$.

**Isolated.** Table 8.19 summarizes the results for the simulation of isolated approaches in each scenario. The application scenarios lead to approximately the same prediction accuracy. The accuracy is high for flow control approaches and becomes increasingly worse for resource provisioning approaches. This is mainly due to inaccuracies in predicting the SLO violations. The prediction of the flow control approaches benefit from the implementation of the synthetic smart devices in the case study system since it is not affected by environmental dynamics to a great extent. Provisioning decisions are executed by the cloud platform. We suspect that the dynamics of provisioning are insufficiently modeled, resulting in a degraded simulation accuracy. Overall, they are at an acceptable level of accuracy that allows the results to be analyzed.

**Coupled.** The results are summarized in table 8.20. The coupling of approaches results in additional dynamics in the system. However, these have little effect on the simulation accuracy. Overall, the prediction accuracy of 32.1 % to 144.25 % is within the acceptable range.

### 8.7.7. Threats to Validity

The presented investigation differs from the investigation in section 8.6 by providing a fixed QoS cost functions. This QoS cost function set is not normalized across each overload scenario but only once for each application scenario. We model application scenarios based on weighting corresponding dimensions of these cost functions. Furthermore, the QoS costs functions are constructed with a quadratic error function, in order to reward approaches, which share the QoS costs across multiple dimensions. Therefore, we refine the external validity to discuss the generalizability of the findings.

| Approach | Scenario | $\delta Q_{total}$ | $\Delta Q_{acc}$ | $\Delta Q_{time}$ | $\Delta Q_{res}$ | $\delta T$ | $\delta Queue_{Delay}$ | $\delta Instances$ |
|---|---|---|---|---|---|---|---|---|
| React | time driven | 291.3 | 0.0 | 5.07 | 0.19 | 0.0 | 520.58 | 18.45 |
| Concurrent | time driven | 142.55 | 0.59 | 0.94 | 0.31 | 14.88 | 302.32 | 19.85 |
| Reg | time driven | 93.53 | 0.0 | 0.03 | 17.59 | 0.0 | 0.0 | 70.1 |
| Rules | time driven | 88.85 | 0.26 | 11.02 | 0.3 | 16.53 | 97.73 | 192.05 |
| CEF | time driven | 75.75 | 0.2 | 5.32 | 0.0 | 10.03 | 71.92 | 0.0 |
| TIF-L AIMD | time driven | 50.4 | 0.69 | 0.77 | 0.0 | 18.13 | 63.55 | 0.0 |
| TIF AIMD | time driven | 36.67 | 1.52 | 0.56 | 0.0 | 24.27 | 38.93 | 0.0 |
| Baseline | time driven | 2.25 | 0.0 | 0.27 | 0.0 | 0.0 | 5.83 | 0.0 |
| React | mixed | 127.05 | 0.0 | 2.31 | 0.21 | 0.0 | 410.85 | 17.8 |
| Reg | mixed | 94.53 | 0.0 | 0.02 | 15.34 | 0.0 | 0.0 | 71.1 |
| Rules | mixed | 79.0 | 0.24 | 4.97 | 0.37 | 18.97 | 95.65 | 213.25 |
| CEF | mixed | 70.47 | 0.22 | 3.53 | 0.0 | 10.15 | 71.78 | 0.0 |
| TIF-L AIMD | mixed | 56.42 | 0.57 | 0.97 | 0.0 | 22.57 | 48.35 | 0.0 |
| TIF AIMD | mixed | 34.9 | 0.69 | 0.7 | 0.0 | 25.75 | 44.92 | 0.0 |
| Concurrent | mixed | 48.67 | 0.65 | 1.85 | 0.13 | 20.35 | 77.08 | 62.4 |
| Baseline | mixed | 2.1 | 0.0 | 0.13 | 0.0 | 0.0 | 5.85 | 0.0 |
| React | accuracy driven | 117.72 | 0.0 | 2.7 | 0.45 | 0.0 | 265.15 | 109.3 |
| Reg | accuracy driven | 89.75 | 0.0 | 0.02 | 14.99 | 0.0 | 0.0 | 65.33 |
| Concurrent | accuracy driven | 85.4 | 0.41 | 1.2 | 0.21 | 20.02 | 93.32 | 52.15 |
| Rules | accuracy driven | 83.2 | 0.47 | 5.35 | 0.15 | 21.23 | 96.65 | 136.63 |
| CEF | accuracy driven | 74.38 | 0.37 | 5.55 | 0.0 | 8.15 | 85.35 | 0.0 |
| TIF-L AIMD | accuracy driven | 49.23 | 1.01 | 1.45 | 0.0 | 19.92 | 63.7 | 0.0 |
| TIF AIMD | accuracy driven | 31.05 | 1.04 | 1.08 | 0.0 | 11.13 | 33.4 | 0.0 |
| Baseline | accuracy driven | 1.85 | 0.0 | 0.12 | 0.0 | 0.0 | 5.68 | 0.0 |
| Reg | cost driven | 91.73 | 0.0 | 0.02 | 31.3 | 0.0 | 0.0 | 67.17 |
| Rules | cost driven | 78.63 | 0.22 | 5.52 | 0.64 | 15.88 | 97.68 | 205.55 |
| Concurrent | cost driven | 77.22 | 0.33 | 2.02 | 0.43 | 27.4 | 151.9 | 48.42 |
| CEF | cost driven | 75.25 | 0.2 | 3.79 | 0.0 | 7.85 | 80.92 | 0.0 |
| React | cost driven | 44.58 | 0.0 | 2.58 | 1.13 | 0.0 | 132.0 | 233.03 |
| TIF-L AIMD | cost driven | 42.13 | 0.47 | 0.47 | 0.0 | 23.1 | 50.95 | 0.0 |
| TIF AIMD | cost driven | 33.48 | 0.58 | 0.35 | 0.0 | 17.27 | 31.77 | 0.0 |
| Baseline | cost driven | 1.85 | 0.0 | 0.11 | 0.0 | 0.0 | 5.48 | 0.0 |

Table 8.19.: Average prediction errors of the QoS costs and the input metrics as percentage difference ($\delta$) or absolute prediction errors ($\Delta$) for the isolated approaches in each application scenario.

| Approach | Scenario | $\delta Q_{total}$ | $\Delta Q_{acc}$ | $\Delta Q_{time}$ | $\Delta Q_{res}$ | $\delta T$ | $\delta Queue_{Delay}$ | $\delta Instances$ |
|---|---|---|---|---|---|---|---|---|
| Concurrent (React, CEF) | accuracy driven | 144.25 | 1.24 | 0.36 | 0.16 | 51.72 | 43.45 | 93.13 |
| Rules (Reg, CEF) | accuracy driven | 92.97 | 0.23 | 5.8 | 0.11 | 15.68 | 98.9 | 110.25 |
| Rules (Reg, TIF AIMD) | accuracy driven | 91.95 | 0.1 | 5.65 | 0.16 | 10.22 | 97.75 | 140.05 |
| Rules (React, TIF AIMD) | accuracy driven | 86.67 | 0.24 | 5.73 | 0.36 | 11.47 | 98.65 | 200.8 |
| Rules (React, CEF) | accuracy driven | 86.0 | 0.64 | 5.63 | 0.06 | 25.57 | 98.4 | 77.25 |
| Concurrent (Reg, CEF) | accuracy driven | 84.08 | 0.24 | 0.0 | 15.8 | 11.65 | 1189.45 | 66.7 |
| Concurrent (Reg, TIF AIMD) | accuracy driven | 73.53 | 0.09 | 0.03 | 12.43 | 7.33 | 1690.55 | 56.55 |
| Concurrent (React, TIF AIMD) | accuracy driven | 61.3 | 1.37 | 1.91 | 0.26 | 48.75 | 165.78 | 45.0 |
| Concurrent (React, CEF) | mixed | 104.85 | 0.67 | 0.07 | 0.3 | 28.42 | 163.63 | 45.72 |
| Rules (Reg, CEF) | mixed | 95.1 | 0.15 | 5.85 | 0.1 | 17.82 | 99.08 | 99.22 |
| Rules (Reg, TIF AIMD) | mixed | 94.3 | 0.06 | 5.73 | 0.18 | 9.47 | 98.77 | 147.9 |
| Rules (React, TIF AIMD) | mixed | 91.03 | 0.07 | 5.49 | 0.14 | 11.35 | 97.4 | 130.52 |
| Rules (React, CEF) | mixed | 87.95 | 0.34 | 5.73 | 0.14 | 23.75 | 98.42 | 110.02 |
| Concurrent (Reg, TIF AIMD) | mixed | 86.47 | 0.0 | 0.0 | 16.03 | 1.57 | 309.4 | 60.83 |
| Concurrent (Reg, CEF) | mixed | 85.88 | 0.05 | 0.0 | 14.74 | 7.52 | 5895.05 | 64.17 |
| Fuzzy (Reg, TIF AIMD) | mixed | 76.33 | 0.05 | 5.69 | 0.72 | 10.2 | 98.35 | 138.9 |
| Fuzzy (Reg, CEF) | mixed | 74.95 | 0.14 | 5.84 | 0.82 | 16.9 | 99.67 | 158.03 |
| Fuzzy (React, CEF) | mixed | 71.63 | 0.65 | 5.8 | 0.42 | 32.95 | 98.78 | 80.15 |
| Fuzzy (React, TIF AIMD) | mixed | 71.38 | 0.3 | 5.62 | 0.68 | 17.15 | 98.38 | 168.35 |
| Concurrent (React, TIF AIMD) | mixed | 32.1 | 0.19 | 0.63 | 0.31 | 10.05 | 197.55 | 13.72 |
| Rules (Reg, TIF AIMD) | time driven | 96.1 | 0.06 | 11.49 | 0.13 | 9.18 | 97.95 | 126.67 |
| Rules (Reg, CEF) | time driven | 95.8 | 0.21 | 11.62 | 0.15 | 18.0 | 98.97 | 132.57 |
| Rules (React, CEF) | time driven | 94.05 | 0.4 | 11.41 | 0.07 | 29.25 | 98.23 | 64.33 |
| Rules (React, TIF AIMD) | time driven | 93.8 | 0.4 | 11.47 | 0.17 | 25.9 | 98.6 | 133.05 |
| Fuzzy (React, CEF) | time driven | 87.9 | 0.25 | 11.56 | 0.68 | 22.18 | 99.03 | 129.13 |
| Fuzzy (Reg, TIF AIMD) | time driven | 87.35 | 0.16 | 11.59 | 0.77 | 16.23 | 99.13 | 156.5 |
| Fuzzy (Reg, CEF) | time driven | 84.38 | 0.2 | 11.71 | 0.88 | 20.27 | 99.68 | 141.48 |
| Concurrent (Reg, CEF) | time driven | 80.17 | 0.18 | 0.01 | 13.46 | 14.2 | 1501.25 | 65.38 |
| Fuzzy (React, TIF AIMD) | time driven | 75.93 | 0.45 | 10.0 | 0.59 | 26.23 | 93.1 | 118.47 |
| Concurrent (Reg, TIF AIMD) | time driven | 73.15 | 0.25 | 0.07 | 13.7 | 17.0 | 5109.27 | 62.3 |
| Concurrent (React, CEF) | time driven | 71.55 | 1.06 | 0.0 | 2.75 | 28.5 | 5692.32 | 43.95 |
| Concurrent (React, TIF AIMD) | time driven | 51.9 | 0.08 | 0.44 | 0.26 | 8.57 | 76.0 | 26.15 |
| Rules (Reg, CEF) | cost driven | 93.3 | 0.19 | 5.76 | 0.08 | 18.88 | 98.5 | 71.85 |
| Rules (Reg, TIF AIMD) | cost driven | 92.2 | 0.11 | 5.73 | 0.25 | 14.38 | 98.35 | 124.8 |
| Concurrent (Reg, TIF AIMD) | cost driven | 84.42 | 0.04 | 0.04 | 29.82 | 7.15 | 11960.5 | 61.73 |
| Concurrent (Reg, CEF) | cost driven | 84.27 | 0.41 | 0.0 | 28.61 | 23.5 | 2965.15 | 64.82 |
| Rules (React, TIF AIMD) | cost driven | 79.8 | 0.2 | 4.88 | 0.22 | 21.73 | 93.88 | 107.22 |
| Rules (React, CEF) | cost driven | 77.0 | 0.33 | 5.56 | 0.75 | 25.75 | 97.72 | 121.33 |
| Fuzzy (Reg, CEF) | cost driven | 70.95 | 0.25 | 5.88 | 1.37 | 23.63 | 99.58 | 121.83 |
| Fuzzy (Reg, TIF AIMD) | cost driven | 69.03 | 0.06 | 5.8 | 1.5 | 11.35 | 99.2 | 148.47 |
| Concurrent (React, TIF AIMD) | cost driven | 68.55 | 0.47 | 1.1 | 0.8 | 19.27 | 395.48 | 39.23 |
| Fuzzy (React, TIF AIMD) | cost driven | 63.48 | 0.46 | 5.71 | 1.58 | 16.68 | 98.48 | 187.53 |
| Fuzzy (React, CEF) | cost driven | 60.25 | 0.62 | 5.82 | 1.41 | 28.57 | 98.78 | 133.85 |
| Concurrent (React, CEF) | cost driven | 36.77 | 0.32 | 0.12 | 0.2 | 15.52 | 63.12 | 23.08 |

Table 8.20.: Average prediction errors of the QoS costs and the input metrics as percentage difference ($\delta$) or absolute prediction errors ($\Delta$) for the coupled approaches in each application scenario.

**External validity.** The investigation aims to generalize the impact of the application scenario on the overload protection quality of different approaches. In practice, there may be applications, which are driven by the combination of qualities, e.g. time and accuracy. However, the investigation allows a general understanding by showing how each application scenario influences the contributions of isolated and coupled flow control approaches on the QoS conformance. We rely on a specific time-varying connectivity pattern. By using optimization heavily, we can not exclude an over-optimization to this connectivity scenario which favors one approach over the other.

**Reliability validity.** To the best of our knowledge, we provided all details needed to replicate the experimental setup. For this reason, we strongly expect the results to be reproducible.

### 8.7.8. Discussion

In this section an investigates the capabilities of (coupled) flow control approaches to maintain the QoS in intensifying overload scenarios for time, accuracy and resource cost driven sensing cloud applications (Goal 6). Overall, it has been shown that isolated approaches are less efficient in coping with different application scenarios since each experience a strong QoS degradation if specific qualities are weighted. All coupling mechanisms have shown to improve the QoS conformance to a strong degree leveraging the contribution of the coupled approaches. The QoS conformance of concurrent coupling depends to a great extent on the emergent behavior of the coupled approaches since some approaches tend to dominate others. A fuzzy rules coupling is able to switch to an isolated operation of the coupled approaches if specific QoS costs are too high. This stabilizes the QoS conformance in all application scenarios making it a good choice for overload protection. The QoS rules coupling depends heavily on fine tuning and can therefore be expected to be less suitable to be used in practice. However, the fine tuning allows to optimize for specific application scenarios resulting in overall low QoS costs. The investigation has strengthen the findings of the investigation of the QoS characteristics on the previous section. Furthermore, it has demonstrated the weaknesses of each overload protection approach in respect to reoccurring application scenarios for sensing cloud applications.

## 8.8. Connected Heating – Use Case 'Predictive Maintenance'

In this section, an investigation of the QoS contribution of the presented flow control approaches on the example of an industry-relevant cloud solution takes place. Therefore it addresses Goal 7 of the validation. The investigation is based on the `Predictive Maintenance` use case of the Connected Heating solution of the Robert Bosch GmbH. In this use case, heating units publishes temperature and pressure data with a high frequency. Since the use case requires data about the environment to analyze it, it demands that the environment is sensed by the heating units with a specific accuracy. Therefore it allows us to quantify the impact of transmission rate adjustments based on the error between the sensed data and the actual environment. Since the presented flow control approaches aim to improve the QoS conformance in overload situations, we introduce an outage and recovery scenario, such that the initial provisioning is eventually insufficient to cope with the connected heating units. We provide QoS cost functions to quantify the contributions of the approaches and compare it to auto-scaling and an accuracy driven collection strategy introduced in [67]. Furthermore, we investigate the benefits and impacts of coupling the approaches with each other.

### 8.8.1. Experimental Design

The outage and recovery scenario is formed by a cloud service with an initial provisioning of 1 instance per microservice and 2000 reconnecting heating units over a time span of 24 minutes. In order to quantify accuracy QoS costs we calculate a target accuracy which emerges for a provided environment model and the sensed data based on a transmission interval declared in the internal documentation. We quantify accuracy degradations by measuring the deviation of the sensed environment to the actual environment. Furthermore, we introduce two QoS cost set functions. Whereas the first captures the sensed accuracy and timeliness violations, the second introduces a measure, which expresses how accurate the measured values are at the time of arrival at the cloud application. We focus on temperature data only and use a homogeneous environment model for all smart devices and do not exploit the correlation of measurements.

**Candidates.** The case study focuses on the following candidate sets, which are compared with each other with regard to different QoS metric:

- **I. Baseline:** As a baseline scenario and as an initial state of the system, we use a statically configured system where each service has exactly one instance and which is not replicated during the experiment. The goal is to observe the behavior of the system without dynamics caused by runtime management approaches. It is therefore used for comparison with the other runtime approaches.
- **II. Isolated Auto-Scaling:** Auto-scalers are the current state of the practice in runtime management of cloud applications. For this reason, an auto-scaling scenario is set up as one of the runtime candidates, in which each individual service is automatically provisioned via an auto-scaler. Furthermore, this setup allows insights into the elasticity of the system, as the elasticity characteristics depend on the cloud platform and the deployed applications.
- **III. Isolated Flow Control (CEF):** This candidate relies on an isolated overload protection approach, which dynamically regulates the transmission rate of the heating units based on performance metrics. The goal is to find out what influence this approach has on the QoS and how it influences the QoS in this process.
- **IV. Isolated Accuracy-Driven (ACC):** This candidate focus on an accuracy driven approach, which adjust the transmission rate according to a target accuracy derived by the change in the current and the last measurement. This state of the art approach

is presented in [67] and included in this case study to investigate its influence on the accuracy and the overall load.

- **V. Flow Control & Accuracy-Driven:** This candidate couples the candidates presented in III and IV, in order to investigate the combination of a time- and an accuracy-driven flow control approach. Since both approaches affect the same mechanism based on different qualities, we use a rule-based coupling that only triggers the overload protection if necessary. Therefore, the strategy candidate aims to dynamically adapt the transmission rate in order to sense the environment with a reduction in the overall load. Since it can induce congestions by increasing the transmission rate or exceeding the available capacity, the congestion avoiding flow control approach aims to mitigate resulting overload situations.

- **VI. Auto-Scaling & Accuracy-Driven:** This candidate couples the candidates presented in II and IV. The goal is to reduce the transmission rate while at the same time adapting the capacity to counteract overload situations. The candidates are operated concurrently.

- **VII. Auto-Scaling & Flow Control:** The candidates presented in II and III are coupled in a concurrent manner. The aim is to provide an additional mechanism to counteract severe overload situations. The time in which necessary resources are provisioned depends on the auto-scaler, the cloud platform and the cloud service. Therefore it may not be always in time, which is why a congestion avoiding flow control approach is utilized.

- **VIII. Auto-Scaling & Flow Control & Accuracy-Driven:** This candidate couples II, III and IV. The idea is to use the accuracy-driven approach to adjust the transmission rate as needed. The required capacity should be provided by the auto-scalers. Therefore, the auto-scalers operate simultaneously with one of the flow control approaches, whereas the flow control approaches are coupled as described in V.

**Sensing Accuracy.** As introduced in chapter 4 the accuracy of measurements by heating units is determined by calculating the percentage error between the measured values and the actual environment. Let $F_t$ be the environment data an $G_t$ the sensed data within at a slotted time $t$ with a total of $T$ discrete time slots. We express the accuracy as the mean percentage error, with 0 % as the best value:

$$accuracy(G) = 100 * \sum_{t=0}^{T} \frac{|G_t - F_t|}{F_t}$$

The accuracy is not dependent on time, as each heating unit sets a timestamp before transmitting a value. The measured temperature curve is reconstructed using this timestamp. In order to determine the accuracy from the point of view of the cloud service, a second error value is calculated, which we call *perceived accuracy.* This error expresses how accurate the measured values are at the time of arrival at the cloud application, i.e. it becomes minimal if there is no processing delay and no measurement inaccuracy.

**Environment Data.** We rely on temperature traces retrieved from `Deutsche Wetterdienst`, which provides hourly temperature data on an open access platform[6]. Based on the coarse granularity of the data, we speed it up by a factor of $2*24*60$, such that each hour is represented by half a minute. Furthermore, we interpolate data linearly in order to retrieve varying data for each second. Whereas this induce more environment changes than to be expected in a real world setup, it allows to shorten the overall experiment duration and to leverage the contribution of accuracy-driven strategies on the QoS. We select all the temperature data of the 31.05.2018 obtained by the weather station in Großenkneten, Lower Saxony. The data is illustrated in figure 8.27.

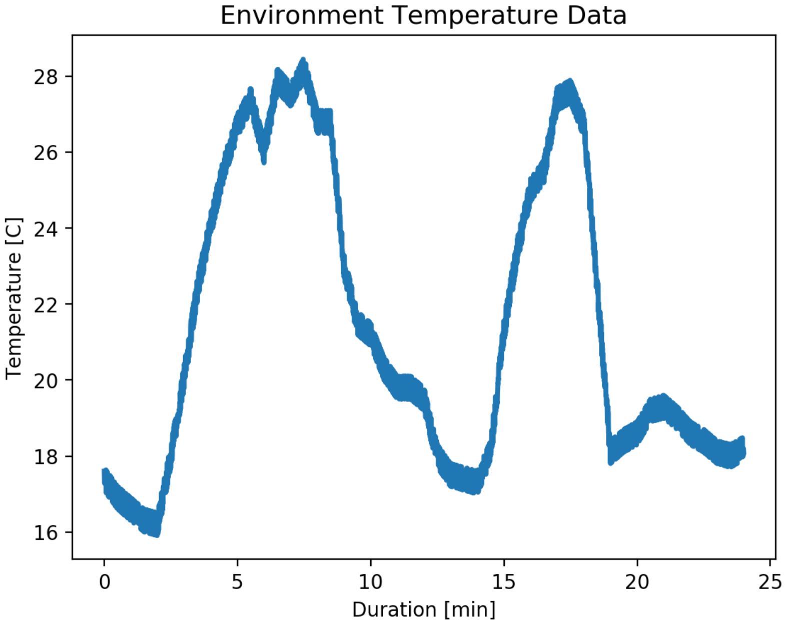


Figure 8.27.: Extracted and prepared Environment Temperature Data from a weather station in Großenkneten, Lower Saxony, during 31.05.2018 - 01.06.2018.

**QoS Cost Functions.** The sensing requirements of cloud services can be driven by different qualities. Whereas some cloud services may require a high accuracy without strict time constraints others demands a high accuracy under strong time constraints. While the former focuses on an analysis of the environment, the latter implies a certain reactivity of the IoT application to changes in the environment. Whereas the use case `Predictive Maintenance` focuses on an analysis, it can not be conclusively answered which cost functions are best suited for evaluation. Therefore, two different sets of cost functions are used in this study as introduced in chapter 4. One set is consistent with the previous investigations and consists of a cumulative cost function for resources, time, and accuracy, such that $\chi_1 = Q_{Accuracy} + Q_{Timeliness} + Q_{Resources}$. In the second set, we combine accuracy and time aspects in a cost function. Here, we look at the accuracy of temperature values in time, i.e. how accurate is the temperature data transmitted at a given time, perceived by

[6] https://opendata.dwd.de/climate_environment/CDC/observations_germany/climate/hourly/

the cloud application. Therefore, we use only a resource and a time-accuracy cost function, which uses perceived accuracy as input, such that $\chi_2 = Q_{PerceivedAccuracy} + Q_{Resources}$. We also introduce a `temporal shift` metric, which quantifies how old the data perceived by the cloud service is at the end time of the experiment. In general, we construct linear cost functions as described in chapter 4.4. We compute the cumulative QoS costs after each experiment. We create a normalized QoS cost function set as described in section 8.6.1. Therefore, we construct a linear cost function for each quality which computes the QoS costs by comparing the measured QoS input metric to its worst value across all candidates. The construction of each cost function is explained below:

- **Accuracy:** While the internal documentation describes a message interval of 15 s, the resulting accuracy of the measurements depends on the environmental changes. Based on the worst measured accuracy $acc_{worst} > 0\,\%$ and the best accuracy of $acc_{opt} = 0\,\%$, we construct the cost function as follows:

$$Q_{Accuracy}(acc_{exp}) = \frac{acc_{exp}}{acc_{worst}}$$

- **Timeliness:** There is no hard constraint on the timeliness described in documentation. Therefore, we rely on the worst measured value in the candidate sets $T_{worst} > 0$ s and a best timeliness of $T_{opt} = 0$ s, such that:

$$Q_{Timeliness}(T_{exp}) = \frac{T_{exp}}{T_{worst}}$$

- **Resources:** Since the actual productively used system is statically provisioned and not operated on a cloud platform, we could not extract any operating costs for the resources. Therefore, we introduce a cost function, based on the highest average number of instances provided accumulated across all cloud services $P_{worst} > 0$ instance, and a best average provisioning of $P_{best} = 0$ instance. Notably, there can not be 0 costs for resources, since there is at least 1 service instance active, per microservice. This results in the following cost function for a given number of instances $P_{exp}$:

$$Q_{Resources}(P_{exp}) = \frac{P_{exp}}{P_{worst}}$$

- **Perceived Accuracy:** This cost function represents the accuracy of the measured data perceived by the cloud service, therefore considering both, the sensing accuracy and processing delays. Based on the worst measured perceived accuracy $acc_{worst}^{perceived} > 0\,\%$ and the best accuracy of $acc_{opt}^{perceived} = 0\,\%$, we construct the cost function as follows:

$$Q_{PerceivedAccuracy}(acc_{exp}^{perceived}) = min(1, \frac{acc_{exp}^{perceived}}{acc_{worst}^{perceived}})$$

**Cloud Service Parametrization.** Based on systematic load tests on the processing service of the connected heating system in a test environment, we obtain an average processing time of 5.8 ms per message with a minor standard deviation of $\sigma = 0.5$ ms. The service exhibits I/O characteristics by waiting for an external system, which results in an average CPU utilization of $CPU = 83\,\%$ in a fully utilized scenario. We were not able to retrieve data on the dispatching service, the heating control. However, we decide to parameterize it with a lower service time of 2 ms and a CPU share of 1. Since the dispatching service waits for the processing service to serve the messages by conducting REST calls, it experiences wait times. In the initial state, each service has 1 active instance. According to internal documents, heating units send messages at a rate of one every 15 s. Therefore, we are taking this value as the basis for the initial transmission rate. This value is set by each connecting device, whereas each device sends at least one message, before being readjusted by flow control approaches.

**Connectivity Scenario.** Whereas connected heating can be expected to have a low variation in the number of connected heating units, we focus on a recovery scenario, e.g. after an outage of the cloud service. This allows to observe the impact of overload protection approaches and the auto-scalers and the combination of both. We rely on a connectivity pattern illustrated in figure 8.28, which we have extracted from a Wikipedia workload peak. We set the number of connected devices up to 2000.

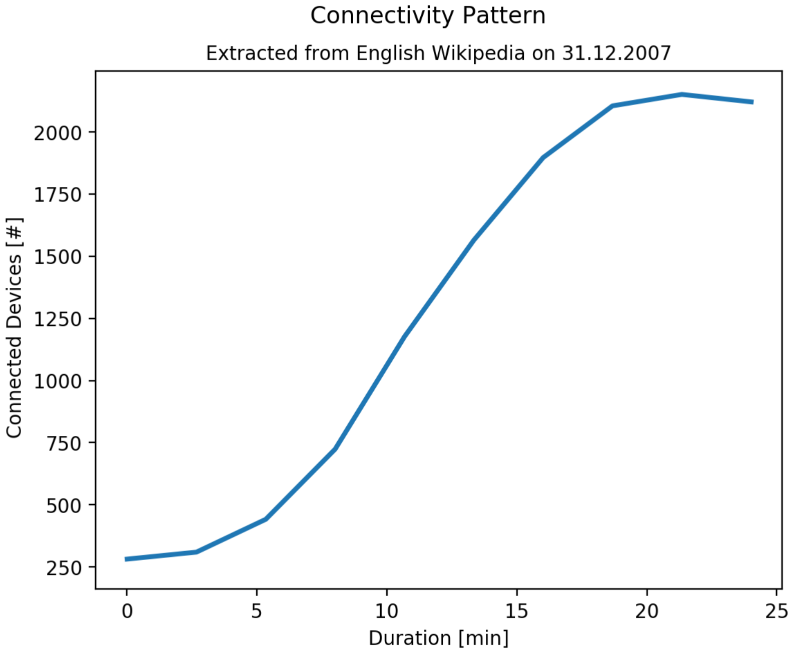


Figure 8.28.: Connectivity Scenario based on a recovery of the IoT system.

**Candidates Parametrization.** Based on the properties of the case study and many experiments on this system, we configure each strategy as follows:

- **II. Auto-Scaling:** Due to the wide distribution of this class of auto-scalers, we rely on threshold based rules auto-scalers, which were introduced as React in [42]. Based on the findings of the performance metrics investigation in chapter 8.4 we select a CPU- and a queue-based auto-scaler:
  - `Heating Control Service Scaler`: Since the heating control service dispatches received messages to the predictive maintenance, it has to wait for its processing. Therefore, we characterize it as a mainly I/O-intensive microservice and configure a queue-based auto-scaler, which monitors the queue length of the consumed message queue as a metric for the services utilization. We set the lower threshold to 0 messages and the upper threshold to 200 messages.
  - `Predictive Maintenance Service Scaler`: We configure the auto-scaler of the predictive maintenance service with a lower threshold of 20 % and an upper threshold with 30 %.

  The cooldown duration after each scaling decision is set for both to 10 s. We set the range of available service replicas to $1 - 50$.

- **III. Flow Control:** Based on the congestion avoidance qualities demonstrated in previous investigations, we select CEF as the congestion avoidance flow control strategy and configure the overload avoidance factor with $k_{avoidance} = 1.05$ and the overload protection factor with $k_{protection} = 0.98$. The initial transmission rate and the upper limit is $T_{upper} = \frac{1}{15\,\mathrm{s}} = 0.07\frac{\mathrm{msg}}{\mathrm{sec}}$. The cooldown duration after each transmission rate adaptation is set to 10 s.

- **IV. Isolated Accuracy-Driven:** We configure the AIMD scheme with an additive increase of $0.005\frac{\mathrm{msg}}{\mathrm{sec}}$ and a multiplicative decrease of 0.8. Furthermore, we configure a target utilization of 1.73 %, since this achieved by the fixed sensing interval defined in the documentation and the environment data used in the case study system.

- **V. Flow Control & Accuracy-Driven:** The rule-based coupling is defined as follows:

$$activate_{CEF} = congestion$$

$$activate_{AccuracydrivenCollection} = \neg congestion$$

  Whereas `congestion` is evaluated by a congestion observer with a window size 10 observing the message queue:

$$congestion = QueueLength \geq 100$$

The presented configuration is used across all case study candidates. Whereas the previous investigations in chapter 8.6 concludes, that the QoS characteristics of a coupled setup is strongly affected by the coupling mechanism and the coupled strategies, we rely, if possible, on coupling mechanism which do not utilize QoS costs. This allows us, to evaluate the QoS conformance in consideration of all experiments. Furthermore, we aim

| Strategy | QoS Costs | | | | Measurements | | |
|---|---|---|---|---|---|---|---|
| | Total | Accuracy | Timeliness | Resources | Accuracy [%] | Queueing Delay [sec] | Instances [#] |
| Auto-Scaling & CEF & ACC | 0.71 | 0.35 | 0.04 | 0.32 | 8.77 | 5.08 | 2.95 |
| CEF | 0.95 | 0.67 | 0.06 | 0.22 | 16.63 | 8.89 | 2.0 |
| Auto-Scaling | 1.1 | 0.07 | 0.03 | 1.0 | 1.75 | 4.3 | 9.17 |
| Auto-Scaling & ACC | 1.18 | 0.51 | 0.04 | 0.63 | 12.8 | 4.97 | 5.76 |
| CEF & ACC | 1.28 | 0.99 | 0.06 | 0.22 | 24.82 | 9.05 | 2.0 |
| Baseline | 1.29 | 0.07 | 1.0 | 0.22 | 1.73 | 141.89 | 1.98 |
| Auto-Scaling & CEF | 1.35 | 1.0 | 0.04 | 0.31 | 24.99 | 5.93 | 2.8 |
| ACC | 1.43 | 0.42 | 0.79 | 0.22 | 10.6 | 111.74 | 2.0 |

Table 8.21.: Measurements and QoS Costs based on the cost function set $\chi_1$ for each strategy in the Connected Heating Case Study.

to improve the comprehensibility of the case study by selecting a specific strategy, e.g. CEF, to use it as a representative of its runtime management class.

### 8.8.2. Q.7 – QoS Contributions in Overload Situation on the Example of an Industry-Based Cloud Application

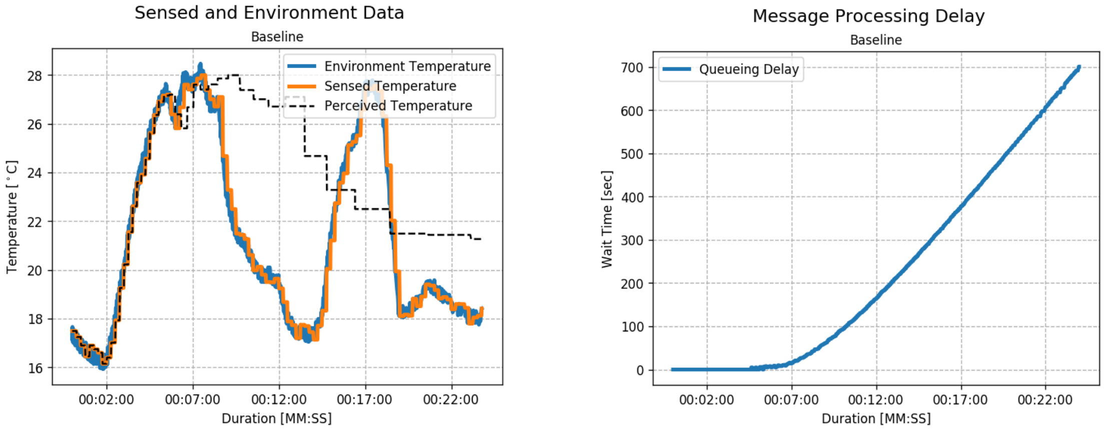
The results of each candidate are summarized in table 8.21 for the QoS cost function set $\chi_1$ and in table 8.22 for the QoS cost function set $\chi_2$.

**I. Baseline.** A static provisioning without further reconfiguration leads to a strong overload situation due to the strong increase of heating units without adjusting the capacity. This results in a high message processing delay caused by accumulated messages as shown in Figure 8.29b. Since the transmission rate is not degraded in this setup it collects and transfers measurements every 15 s which leads to a very high sensed accuracy of 1.7 %. However, based on the high message processing delay measured values arrive at the cloud service with a considerable temporal shift. At the end of the experiment, the measurement points are already delayed by 14 min, which strongly affects the perceived accuracy. Figure 8.29a shows the sensed and perceived accuracy for the first connected heating unit. The perceived accuracy degrades to 15.43 %. In terms of the QoS cost functions, the baseline is in a mid range. In $\chi_1$ its costs mainly consist of timeliness violations. In $\chi_2$ its costs are mainly based on the perceived accuracy. Overall, the static provisioning is not suitable for

| Strategy | QoS Costs | | | Measurements | | |
|---|---|---|---|---|---|---|
| | Total | Perc. Accuracy | Resources | Perc. Accuracy [%] | Instances [#] | Temporal Shift [sec] |
| Auto-Scaling & CEF & ACC | 0.64 | 0.32 | 0.32 | 7.98 | 2.95 | 273 |
| ACC | 0.68 | 0.46 | 0.22 | 11.45 | 2.0 | 533 |
| CEF | 0.77 | 0.56 | 0.22 | 13.89 | 2.0 | 51 |
| Baseline | 0.83 | 0.62 | 0.22 | 15.43 | 1.98 | 825 |
| CEF & ACC | 0.89 | 0.68 | 0.22 | 16.89 | 2.0 | 198 |
| Auto-Scaling & CEF | 1.04 | 0.74 | 0.31 | 18.45 | 2.8 | 64 |
| Auto-Scaling & ACC | 1.19 | 0.56 | 0.63 | 14.11 | 5.76 | 289 |
| Auto-Scaling | 1.36 | 0.36 | 1.0 | 9.0 | 9.17 | 563 |

Table 8.22.: Measurements and QoS Costs based on the cost function set $\chi_2$ for each strategy in the Connected Heating Case Study.

coping with an outage and recovery scenario, since the messaging middleware is filled with over 240 000 messages at the end of the experiment.



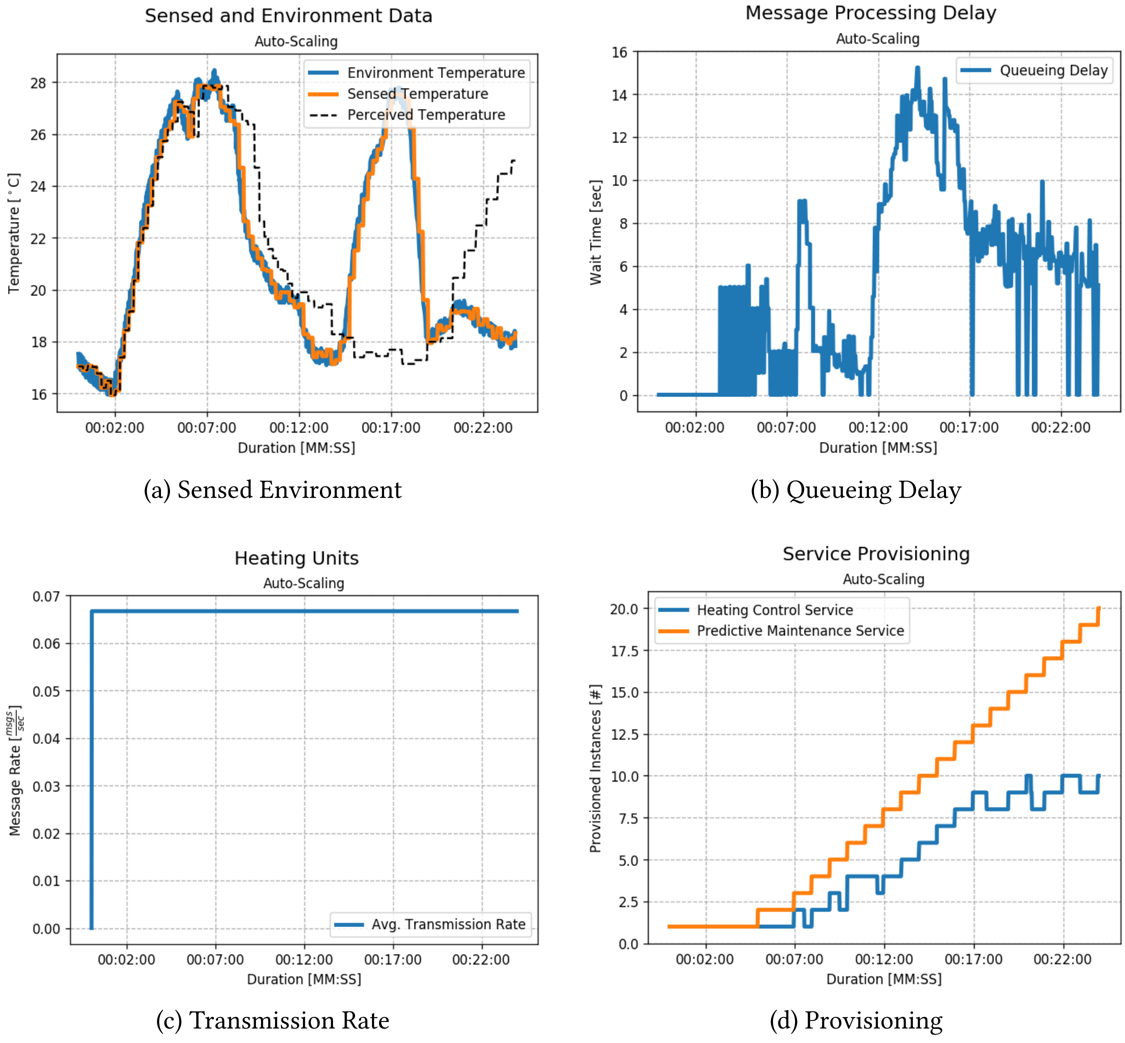
(a) Sensed Environment

(b) Queueing Delay

Figure 8.29.: Baseline. Sensed and perceived environment and message processing delay during the experiment.

**II. Isolated Auto-Scaling:** As shown in figure 8.30 the auto-scalers manage to counteract the strong increase in active heating units by providing resources. However, the provisioning speed of the auto-scalers is challenged by the cloud platform and the provisioning mechanism. A CPU update occurs approximately every 30 s and the provisioning of an instance takes more than 45 s. The configured cooldown duration of 10 s is negligible. A delay due to accumulating messages can not be avoided completely. While messages are delayed by only 4.3 s on average, this has a strong effect on the perceived accuracy. The sensed accuracy, as in the baseline, is 1.7 %, but the perceived accuracy is 9.0 %. Whereas this is a significant improvement over the baseline, it is achieved on the expense of the provisioning many resources, on average 9.17 instances. In $\chi_1$, isolated auto-scaling performs very well with a total cost of 1.1, because it is primarily resource based and there are minor time or accuracy costs. However, due to the cost function construction, the total cost is in favor of auto-scaling, since the baseline is penalized with fixed resource costs of 0.22. In $\chi_2$ an opposite effect can be observed and auto-scaling has the worst score of 1.36, since although perceived accuracy leads to low costs, resource costs dominate, since both are equally weighted. Overall, auto-scaling is a suitable way to deal with such a scenario. While it is challenged by the speed in resource provisioning, it eventually manages to provide enough capacity to cope with the load.

**III. Isolated Flow Control:** Figure 8.31 shows the results for CEF. It shows that above a certain number of connected heating units, the pre-configured message interval leads to a load that exceeds the capacity of the cloud service. Therefore, the first congestion



(a) Sensed Environment

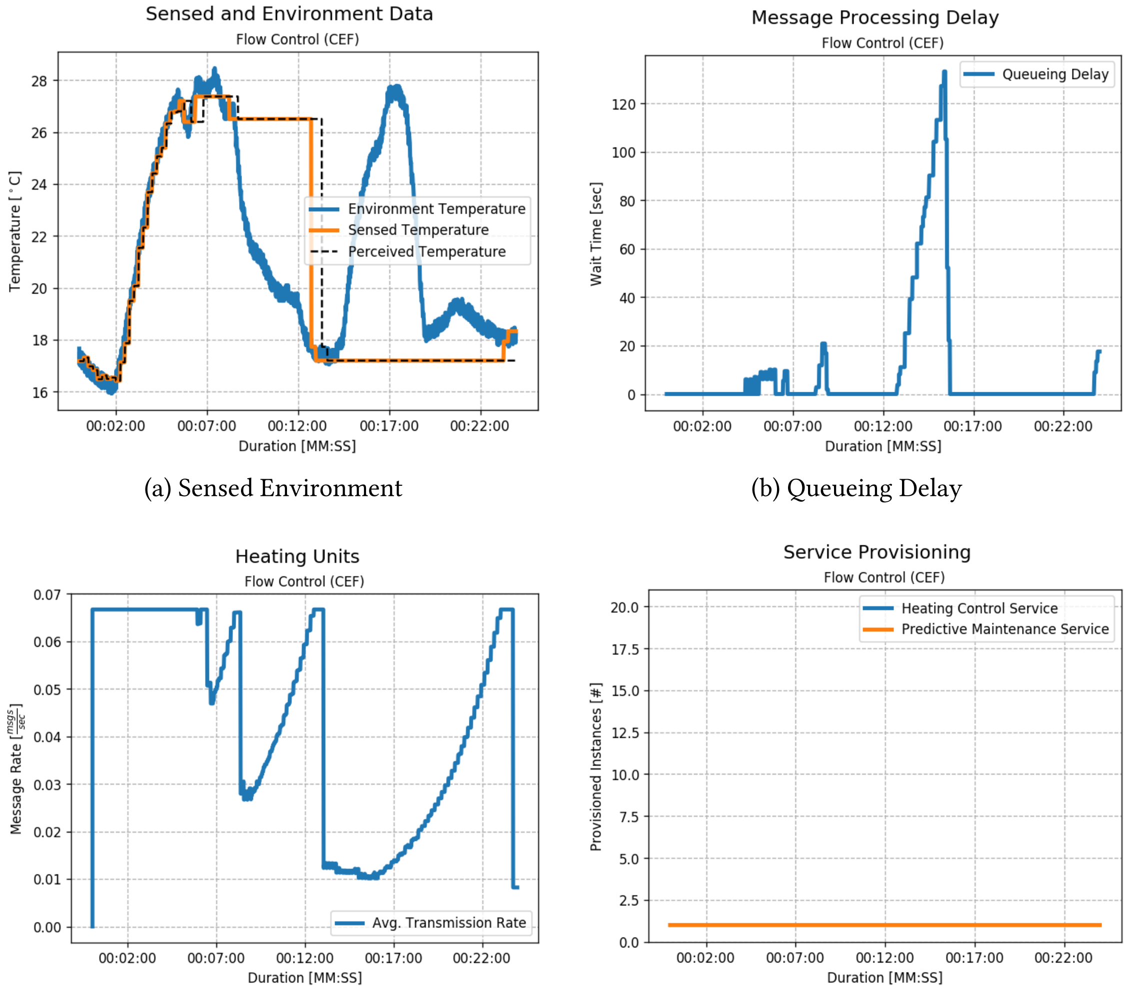
(b) Queueing Delay

(c) Transmission Rate

(d) Provisioning

Figure 8.30.: Isolated Auto-Scaling. Measurements during the experiment.

avoidance cycle is triggered at about 6 min. Based on the capacity model of the process, the transmission rate is reduced significantly and then, when the messaging middleware has recovered, the transmission rate is increased multiplicatively. Since the load increases continuously, a point is reached where the configured rate can not be maintained. The cycles become longer and longer, because the transmission rate has to be degraded more strongly due to the higher number of heating units, while the capacity remains the same. Eventually, with the same number of units, the process will level off at a specific cycle. Due to the increasingly strong degradation of the transmission rate, fewer measurements are performed resulting in sensing intervals which extend several minutes. Overall, the approach leads to a strong reduction of the sensed accuracy, since the environment is only roughly captured by the few measurement points. On average the sensed accuracy is 16.63 %. Since the approach is able to avoid a queueing delay to a large extent, the average queueing delay is 8.89 s. Due to the cycles, however, it exceeds the queueing delay from auto-scaling. The perceived accuracy for the first device is 13.8 % and is therefore close to the sensed accuracy. The reason for this is that there is hardly any temporal delay, which means that the measurement points are inaccurate overall, but in time. In $\chi_1$, CEF has the second best result, with QoS costs of 0.95. These are so low because the time costs are negligible and the accuracy costs are in the middle range. In $\chi_2$, it reaches a total cost of 0.68, which is also relatively low. The good result is due to the mediocre accuracy at low resource costs. Overall, this approach is suitable for keeping the provisioning costs low. However, the congestion avoidance cycle leads to a certain pattern of perception that can result in not noticing important changes in the environment. Therefore, the low total costs within this experiment should be treated with caution.



(a) Sensed Environment

(b) Queueing Delay

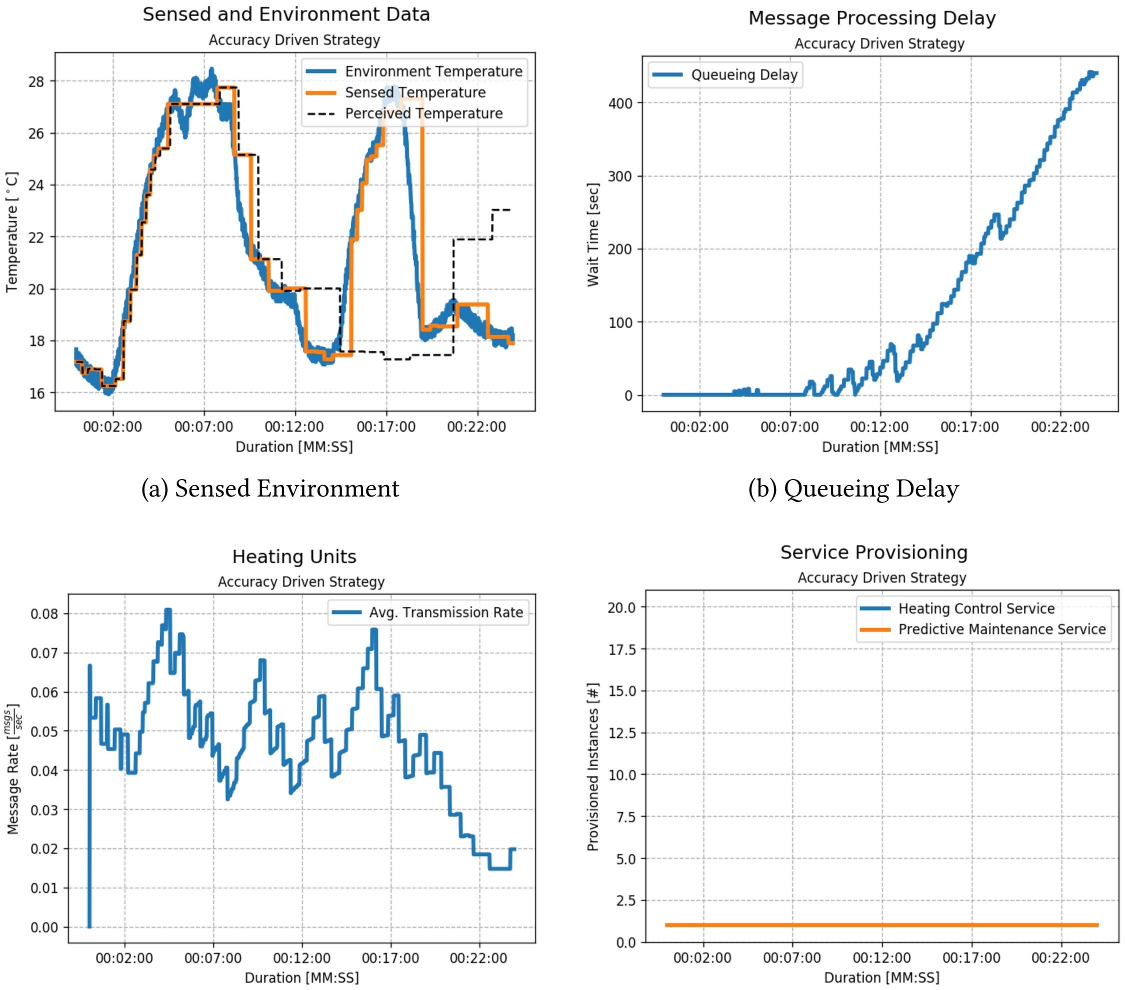
(c) Transmission Rate

(d) Provisioning

Figure 8.31.: Isolated CEF. Measurements during the experiment.

**IV. Isolated Accuracy-Driven:** Given a target accuracy, the accuracy driven approach adapts the transmission interval based on the difference between the current and previous measurement. While the target accuracy is configured to perceive the environment with an accuracy of 1.73 %, the accuracy actually achieved depends on both the environmental changes and the parametrization of the approach. In this experiment, the approach results in a sensed accuracy of 10.6 %. Since the approach potentially reduces the number of measurements, especially in episodes with few environment changes, a lower load on the cloud service can be assumed. However, it still exceeds the cloud services' capacity resulting in a very high queueing delay of 112 s. Nevertheless, there are significantly fewer messages produced, which is reflected in the average transmission interval. While it is 15 s for the baseline, it is 22 s for the accuracy-driven approach. The transmission rate adaptations illustrated in Figure 8.32c show how the approach uses the AIMD scheme to adjust the rate as the environment changes. Figure 8.32a shows that, overall, temperature values are measured — for the first device — with a high degree of accuracy of 3.96 %. However, they arrive with a delay, resulting in a degraded perceived accuracy of 11.45 %. Using the cost function set $\chi_1$, the approach results in a total cost of 1.43. This is the worst result, because although the resource costs are low, the time costs are high and the accuracy is only slightly above average. In $\chi_2$ the approach takes the second highest place, since the perceived accuracy is very high despite the temporal shift. All in all, the findings from [67] are confirmed in this experiment. It is an approach that can achieve a high accuracy and reduces the number of measurements needed. In contrast to the other approaches, the focus is not on load and capacity but on cost reduction while maintaining a target accuracy.

**V. Flow Control & Accuracy-Driven:** Figure 8.33 shows the results of a coupling of CEF and ACC. The interaction of the two approaches is shown by the transmission adjustments. In case of sudden reductions, an overload situation has occurred, which activates CEF. Especially towards the end, CEF dominates strongly, because it is a very intensive overload situation. This results in a complex interaction between the approaches, which is primarily determined by the rule-based coupling mechanism. The coupling uses a simple threshold based on the queue length to determine the overload state. For both ACC and CEF the coupling mechanism is transparent, which means that both approaches can not react to it. The adaptation behavior is generally more unsteady than in isolated CEF or isolated ACC. This is due to the fact that the trigger activates ACC even before CEF considers the overload situation as solved. ACC does not use a multiplier to smoothly recover the transmission rate. Instead, it uses an additive component that causes sudden increases, e.g. at 7.5 min. This sudden increase causes the queue to fill up with messages, resulting in an reactivation of CEF, which tries to recover the transmission rate through the multiplier component. Overall, there is little harmonic effect, which is also expressed in the fact that the message interval is significantly reduced by over 15 s compared to an isolated CEF with 37.8 s. This affects the sensed accuracy, which is almost the worst at 24.82 %. However, the perceived accuracy is much better with 14.11 %, because CEF prevents delays and because of the smaller cycles and more frequent short-term overload situations, allowing to sense more evenly distributed data points. Using the cost function set $\chi_1$ it achieves total costs of 1.18 and is thus in the middle field, since hardly any time and resource costs are incurred and the total costs are therefore primarily composed of the inaccuracy. In $\chi_2$ the total cost is 0.89, which also consists of the average perceived accuracy and the static provisioning.



(a) Sensed Environment

(b) Queueing Delay

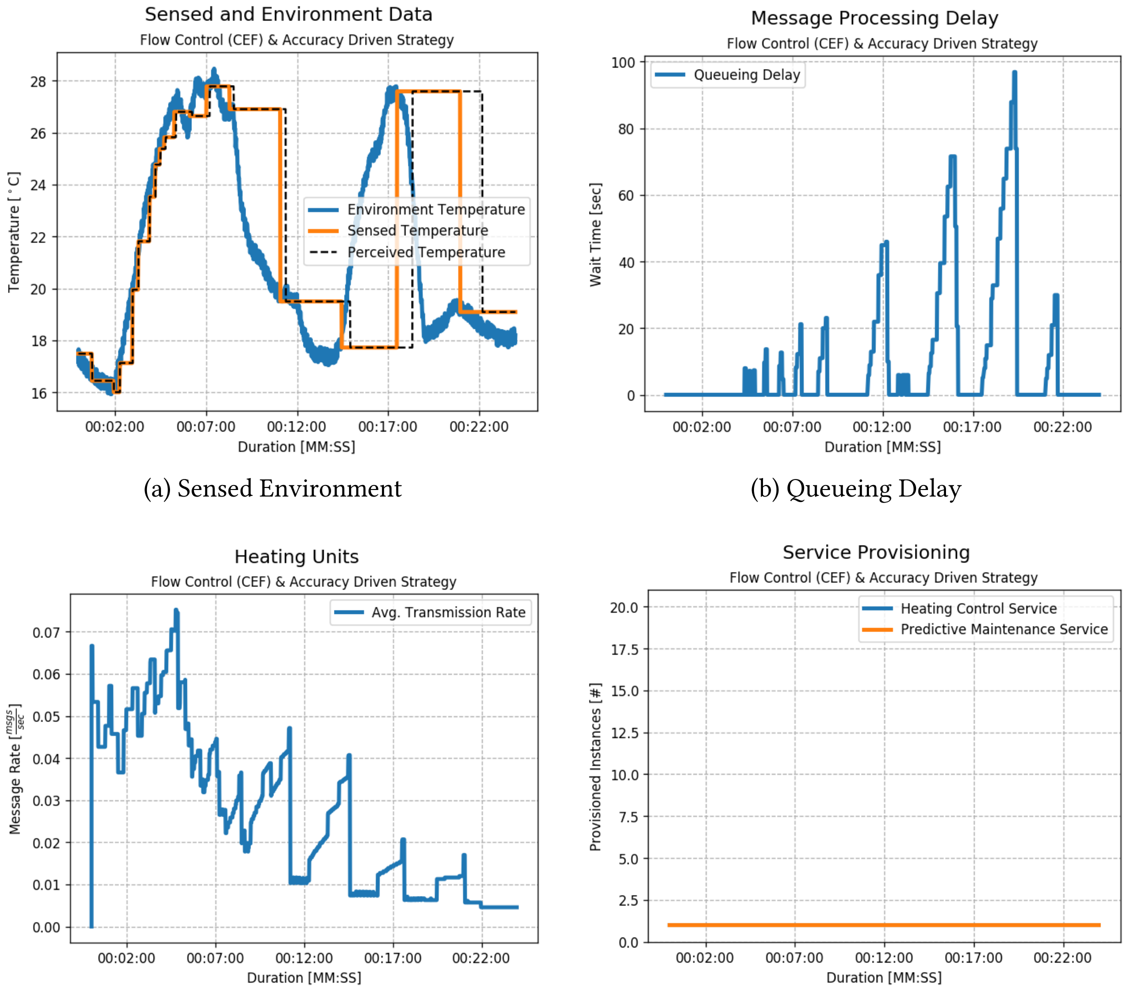
(c) Transmission Rate

(d) Provisioning

Figure 8.32.: Isolated Accuracy-Driven. Measurements during the experiment.

All in all, the combination of these two approaches is suitable, unless a constant overload situation — as in this experiment — has to be assumed.

**VI. Auto-Scaling & Accuracy-Driven:** Figure 8.34 shows the results for a concurrent operation of the auto-scalers and ACC. Since ACC reduces the overall load by adjusting the transmission rate, the overload situation is mitigated. As with the isolated ACC, the average message interval is 22 s, which corresponds to a load reduction per device of roughly 30 %. The measured accuracy is 12.8 %, which is a slight degradation compared to the isolated ACC, which achieves 10.6 %. However, it can be assumed that a coupling with auto-scalers does not influence the behavior of ACC. Therefore, we conclude, that dynamics in monitoring and effecting are responsible for this. Due to the reduction in the load by ACC, fewer resources are required overall, which is shown by an average of 5.76 provisioned instances, compared to 9.71 in isolated auto-scaling. Both share a similar average queueing delay, which differ in its progression. In isolated auto-scaling, the high



(a) Sensed Environment

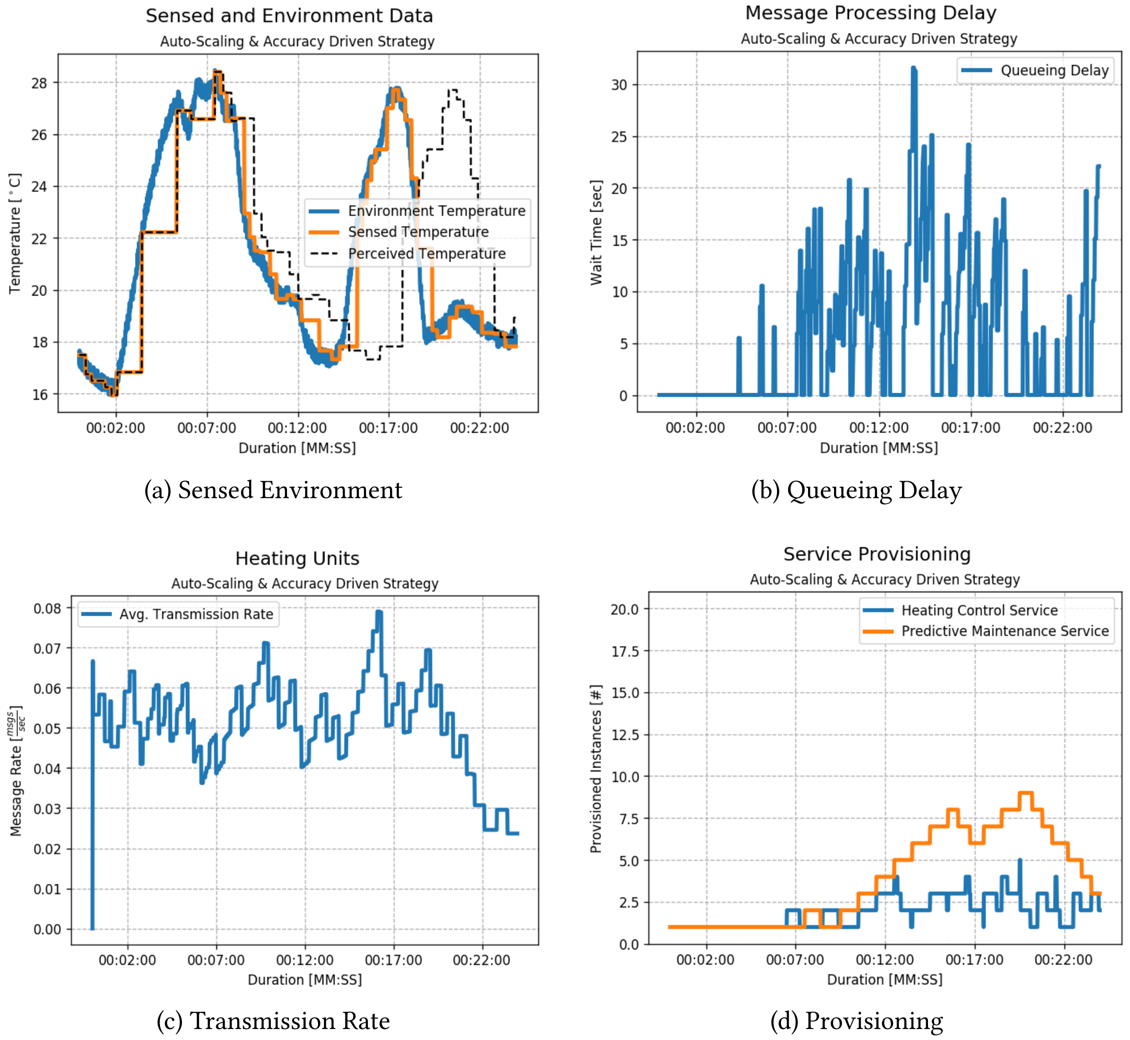
(b) Queueing Delay

(c) Transmission Rate

(d) Provisioning

Figure 8.33.: CEF & Accuracy-Driven. Measurements during the experiment.

provisioning results in only slight delays of less than 8 s towards the end of the experiment. Nevertheless, the time delay at the end of the experiment is with 563 s significantly higher than in coupled operation with only 289 s. Nevertheless, this has hardly any effect on the perceived accuracy, e.g. 14.11 % for coupled operation, while it is 9 % percent for isolated auto-scaling. However, it should be noted that the perceived accuracy depends on the message processing delay, sensed accuracy and actual environmental changes, and the combination of these factors seems to have been advantageous for the auto-scaler in this time-limited experiment. Based on the first cost function set $\chi_1$, this coupling is in the upper range with 1.28 and feeds the costs primarily from accuracy and resources. In the set $\chi_2$, the coupling leads to high costs based on the inferior perceived accuracy. Despite these results, a coupling of these approaches is quite conceivable, since it dynamically reduces the load, yet achieves a high degree of accuracy and can thus act as a supplement to resource provisioning.



(a) Sensed Environment

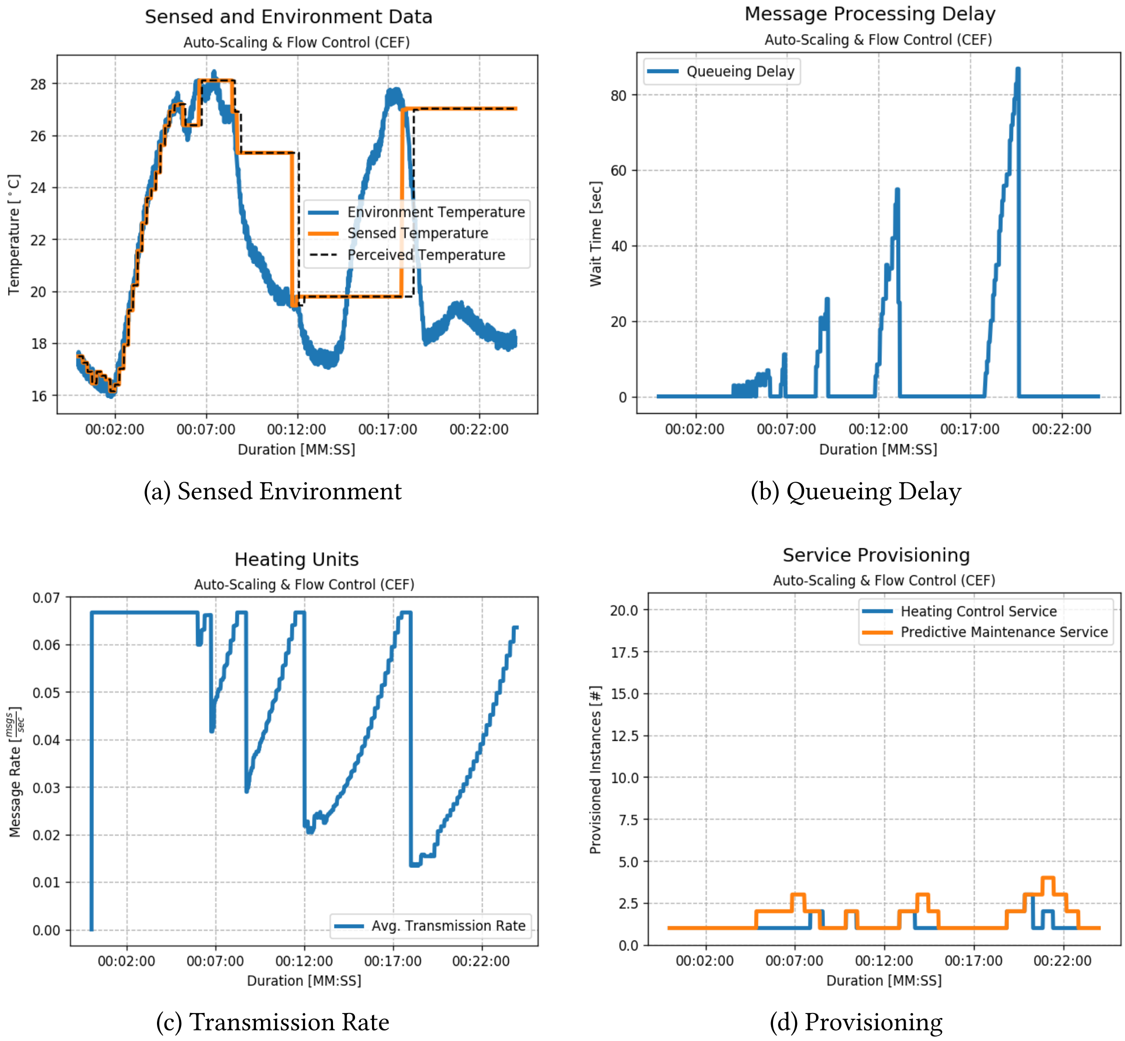
(b) Queueing Delay

(c) Transmission Rate

(d) Provisioning

Figure 8.34.: Auto-Scaling & Accuracy-Driven. Measurements during the experiment.

**VII. Auto-Scaling & Flow Control:** Figure 8.35 shows the adaptation behavior of the concurrent coupling of the auto-scalers and CEF. Compared to the isolated CEF, congestion avoidance cycles have become shorter due to an increased capacity by the auto-scalers. The capacity estimation of the approach is able to react to it by decreasing the transmission rate to a lesser degree. While the auto-scalers are able to react to the overload situation, their effectiveness is reduced by the high congestion avoidance efficiency of CEF. Thus, with an average of 2.8 provisioned instances, significantly fewer resources are used than in an isolated auto-scaling with 9.17 instances. CEF dominates the runtime management, which is also evident from the average message interval. On average, messages are sent every 20 s, which is only slightly below the average of an isolated operation with 23 s. Nevertheless, this setup provides a high congestion avoidance, since messages do not experience a particularly high delay. Thus, at the end of the experiment the measurement points arrive with a delay of less than one minute, which is a significant improvement compared to isolated auto-scaling. Resource provisioning via the auto-scalers induce dynamics that reduce the adaptation quality of CEF. This results in an overall lower measured accuracy of 25 %, while the perceived accuracy is around 18.45 %. Both values are significantly worse than in the isolated approaches. In isolated auto-scaling, the measured accuracy is very high with 1.75 %, while the perceived accuracy is good at 9 % despite a time offset of almost 9 minutes. Both in cost function set $\chi_1$ and $\chi_2$ the strategy results in relatively high QoS costs. The reason for this is an antagonistic coupling effect degrading the adaptation quality of both approaches. However, the temporal shift is small and thus potentially a high perceived accuracy can be achieved in overload situations. As pointed out in the previous investigation in chapter 8.6, CEF dominates in a concurrent coupling. Therefore, it can be assumed that better results can be expected with a rule-based coupling or other flow control candidate.



(a) Sensed Environment

(b) Queueing Delay

(c) Transmission Rate

(d) Provisioning

Figure 8.35.: Auto-Scaling & Flow Control. Measurements during the experiment.

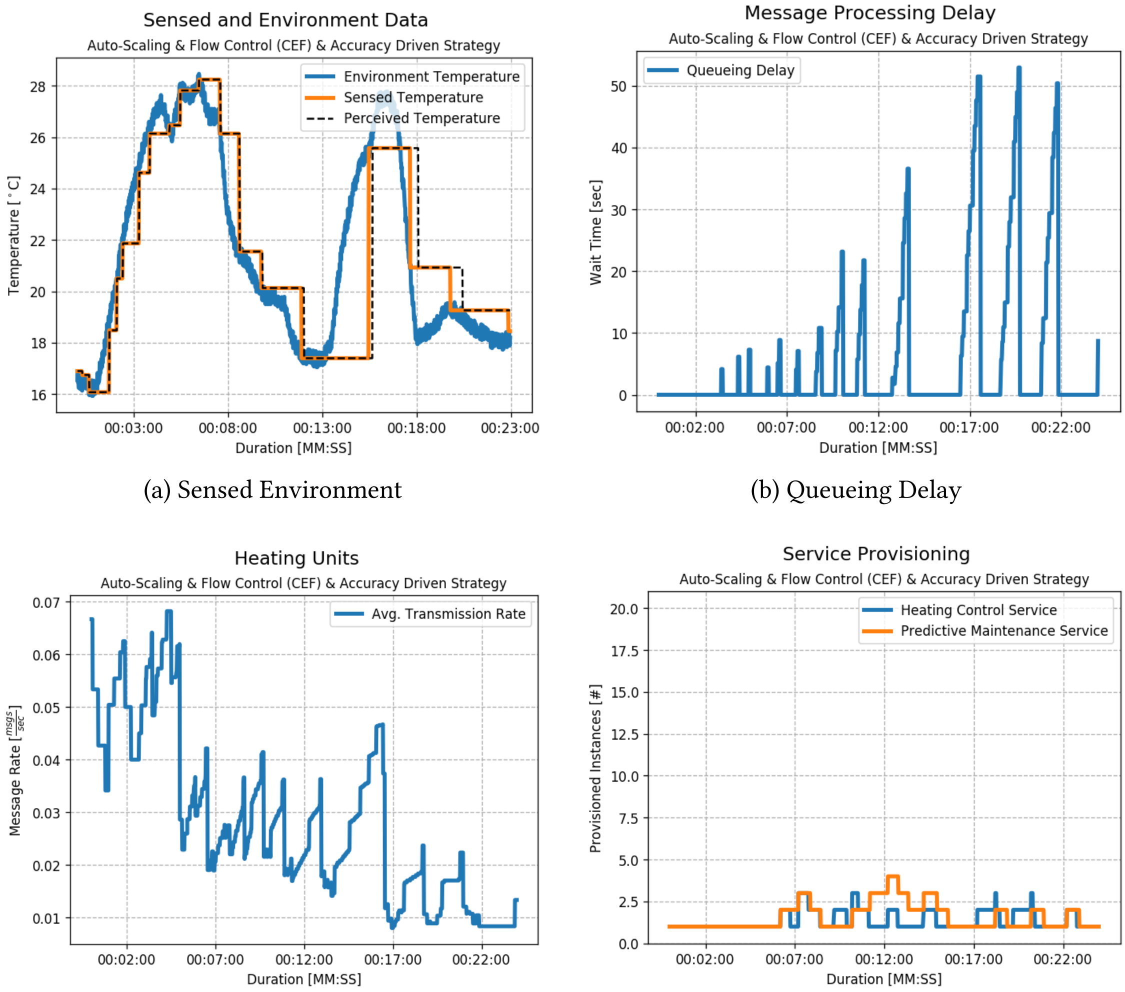
**VIII. Auto-Scaling & Flow Control & Accuracy-Driven:** Figure 8.36 shows the measurements during the experiment. In contrast to a coupling of CEF and ACC a significantly higher accuracy of 8.77 % is achieved, which is based on an increased capacity by resource provisioning. The accuracy also exceeds that of the coupling of CEF with auto-scalers, since we observed an antagonistic effect on the accuracy. This is mainly due to the additional coupling with ACC, which has already shown in the coupling with auto-scalers that it is able to reduce the overall load with a moderate loss of accuracy. All in all, this coupling is convincing in multiple dimensions. However, transmission rate adaptations based on CEF result in a strong transmission rate adaptations and a light resource provisioning. The messages experience only minor processing delays, such that the accuracy perceived by the cloud service is very high at 7.98 %. The provisioned resources are with with 2.95 instances in the midfield and slightly above the auto-scaler and CEF coupling. Overall, the coupling achieves low QoS costs in both cost function sets $\chi_1$ and $\chi_2$. The reason is, that the sensed and the perceived accuracy is relatively high with only minor resource costs. Therefore we consider this as the most promising approach for runtime managing sensing cloud applications and deem it as a significant improvement over the other approaches. We assume that there is room for optimization, since, as already observed in the previous experiments, a coupling with CEF dominates auto-scaling.

### 8.8.3. Threats to Validity

The threats to validity of the case study are discussed below.

**Internal validity.** As in the ShapeShifter case study presented in chapter 8.6, we derive the input for the QoS functions by monitoring the adjusted transmission rate, the number of provisioned resources and the queueing delay. In contrast to this investigation, we do consider an environmental model in order to quantify the accuracy degradations by the transmission rate reduction. Since the calculation is based on a provided environment model, we can exclude any influencing third factor on the accuracy. Furthermore, this case study system consists of multiple cloud services, which make it more challenging to exclude a third factor, e.g. based the additional communication layer, which affects the overall processing rate. This may affect the resource and timeliness cost function by changes in the capacity. However, since the experiments are running over a longer period of time and we execute the experiments on an isolated Kubernetes Cluster, we deem the extent of it as negligible. Nevertheless, we have noticed, that the resulting processing rate is below the expected processing rate.

**External validity.** By preseting a case study, which utilizes environment data to quantify accuracy violations, we expect an increased generalizability for sensing cloud applications. We deem the findings as relevant for cloud applications, which may experience strong connectivity variations. While the study focuses on overload situations, the approaches examined here and their effects are not limited to these. In particular, the coupling of auto-scaling with accuracy-driven collection approaches shows the possibility of dynamically reacting to changes in the environment and thereby provisioning resources.



(a) Sensed Environment

(b) Queueing Delay

(c) Transmission Rate

(d) Provisioning

Figure 8.36.: Auto-Scaling & Flow Control & Accuracy-Driven. Measurements during the experiment.

**Construct validity.** While the system in the study is based on a real environment, the actual cloud application is minimalistic, with a deterministic response time per message. However, since the interaction of the two cloud services creates dynamics that can also be observed in real systems, we consider the setup sufficient to represent such an application. In addition, as in previous investigations in this thesis, the smart devices do not experience latencies, which means that approaches that adjust the transmission rate have an advantage over auto-scaling. We assume that the system would react more slowly with latencies, without negating its fundamental effect. Whereas many smart devices are connected in wireless networks, the heating units are installed using the cable connection of the customers home. Since the customers are regionally distributed in Germany, we assume that there is only a minor latency based on the distance and the stable network. Furthermore, we have utilized a single environment model for all smart devices, which does not hold true in practice. Since the smart devices connect to a different point of time, they perceive the environment with a different accuracy.

**Reliability validity.** We assume that the experimental design of the case study has been described in sufficient detail. Therefore we expect the results to be reproducible.

### 8.8.4. Discussion

This chapter presented a case study based on an industry-relevant cloud solution. This involved intelligent heating control units that continuously transfer measured values over messaging middlewares to a cloud service. Based on the thesis' focus on overload protecting approaches, we investigated an outage and recovery scenario. We investigated auto-scaling, a congestion avoiding flow control and finally an accuracy driven collection approach. Each approach was deployed in isolation and in coupling with the other approaches. We were able to show that a coupling of all three approaches yields the most benefits, since it keeps timeliness violations to a minimum and is able to significantly reduce the overall load with a comparatively low loss of accuracy. Each approach on its own has clear disadvantages. Runtime management with CEF leads to strong cycles, which result in erratic sampling. ACC does not consider time aspects and is therefore unsuitable for overload situations, but can significantly reduce the total load depending on changes in the environment through dynamic adaptation. Popular auto-scaling techniques are too sluggish to cope with the rapidly increasing load, which ultimately leads to overprovisioning with a high time violation. We consider the findings of the case study to be meaningful enough to be valid for other sensing cloud applications.

## 8.9. Characteristics of TCP-inspired Flow Control

In this section, we investigate the characteristics of the TIF AIMD in a distributed setup. We focus on the AIMD scheme based on its superior efficiency and fairness in distributed setups compared to other increase/decrease schemes [22]. We conduct the investigation on the Bosch IoT Cloud using the ShapeShifter case study.

### 8.9.1. Experimental Design

We provision the microservice statically and let the smart devices connect over time using a connectivity pattern. We capture the fairness and obtained transmission rate. We compare the results of a distributed and centralized setup. We adapt the ShapeShifter-IoT case study to propagate the binary congestion state to smart devices. Each smart device utilizes a dedicated AIMD-based request scheme. Each smart device receives the congestion state after transmitting data by piggy-backing.

**AIMD Scheme.** The AIMD scheme is configured with the additive increase parameter $a = 0.1\frac{msgs}{sec}$ and the multiplicative decrease parameter $b = 0.9$. The transmission rate adaptation is limited to a minimum of $0.2\frac{msgs}{sec}$ to ensure that an adaptation step occurs within a foreseeable period of time in a severe overload situation.

**Congestion Observer.** The congestion observer to monitors the queue length with a moving average size of 10 and deems the cloud solution as overloaded if a threshold of 10 is exceeded. The congestion observer retrieves every second an update from the infrastructure.

**Connectivity Scenario.** We rely on a connectivity pattern, which has an increase, a steady and a decrease phase, ranging from 0 to 12 devices.

**Fairness.** We measure the fairness in a distributed setup by applying Jain's Fairness Index, which is defined in [43] and is applicable to any resource sharing problem. The index is bounded between 0 and 1, whereas 1 means absolute fairness.

**Supported Transmission Rate.** In order to illustrate the quality of adaption decisions, we compare the transmission rate adaptations to a theoretical supported transmission rate. Since we control the processing rate of the cloud solution, it is not subject to strong variations resulting in a nearly deterministic processing rate $c$. The optimal transmission rate $T_{Opt}$ is based on the processing rate, the number of devices $N_D$ and an upper boundary $T_{Sat}$ as follows: $T_{Opt} = min(\frac{c}{N_D}, T_{Sat})$

**Experiment Setup.** We set the experimental duration to 10 minutes and set a fixed processing rate of $10.0\frac{msgs}{sec}$ and an initial transmission rate and upper boundary of $T_{init} = T_{sat} = 1.0\frac{msgs}{sec}$.

### 8.9.2. Q.8.1 – How does varying load affect the fairness of adaptations in a distributed setup?

Figure 8.37 shows the transmission rate of each smart device during the experiment. Figure 8.38 shows the achieved fairness at each point of time and on average during the experiment.

With an average fairness of 0.95, the AIMD scheme maintains its efficiency and fairness characteristics in distributed setups. The results show, that the approach is vulnerable to a changing number of connected devices, especially if the number of devices is increasing. Since a smart device starts with a certain initial transmission rate, it requires some round-trips to let itself and the other devices converge towards a fair share of the processing capacity of the cloud application. If the number of connected devices remains steady, the fairness converges quickly towards 1.0, demonstrating absolute fairness.

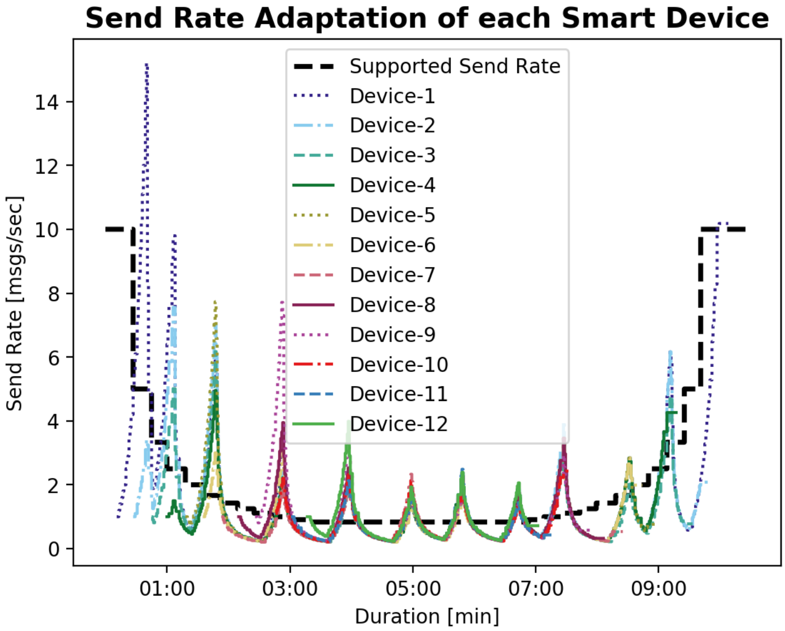


Figure 8.37.: Adaptation behavior for each smart device in a distributed setup. It achieves fairness in a steady state and converges to the supported transmission rate.

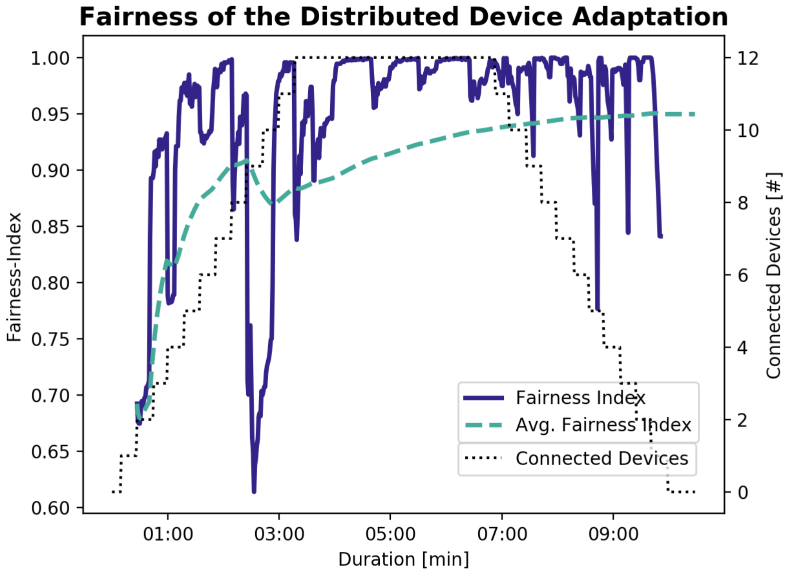


Figure 8.38.: Average and current Jain's Fairness Index. The fairness increases greatly for a steady state with a fixed number of devices. The fairness is especially vulnerable to a changing —and especially increasing— number of connected devices.

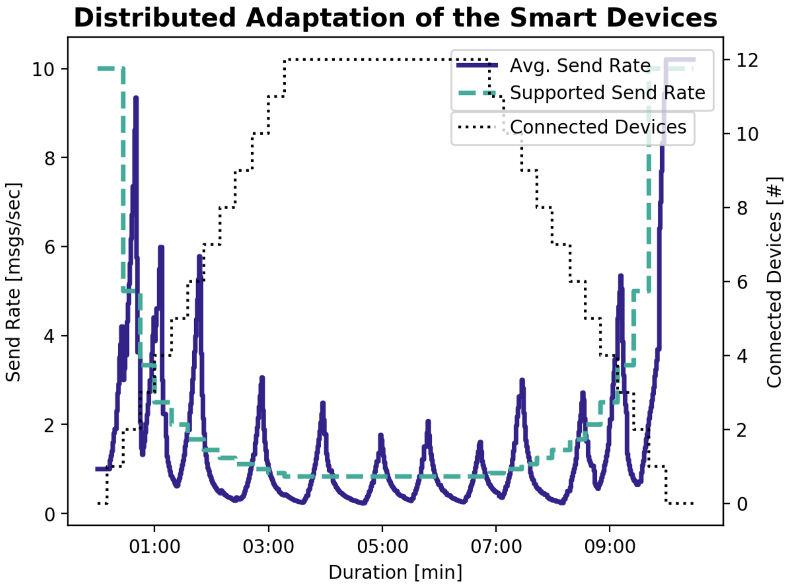


Figure 8.39.: Adaptation behavior and number of the connected devices during the experimental run in a distributed setup.

### 8.9.3. Q.8.2 – How does a distributed setup affect the adaptation quality?

Figure 8.40 shows the adapted transmission rate, the supported transmission rate and the number of connected devices during the experiment in a centralized setup. Figure 8.39 shows the adaptation behavior in a distributed setup.

**Centralized Setup.** The approach achieves an average service utilization of 65 %, a queueing delay of 6.2 secs and a queue length of 24.5 messages. The service utilization is strongly degraded by a substantially lower initial transmission rate value compared to the processing rate of the cloud application, resulting in the need of many round-trips to converge to this value. However, this effect is mitigated by the increasing number of devices. In the steady state in the middle section of the experiment it achieves an average service utilization of 87.1 %.

**Distributed Setup.** The approach achieves an average service utilization of 70.1 % during the experiment and 84.0 % in a steady state. However, by the degraded adaptation quality, congestions are on average more severe, which is reflected by an average queueing delay of 6.9 secs and an average queue length of 32.3 messages.

**Discussion.** Since each smart device adapts itself its transmission rate eventually converges to the supported transmission rate of the cloud application. Connecting devices start with the preconfigured initial transmission rate and require some round trips to achieve a fair send rate in accordance to already connected devices. This degrades the overall quality of the adaptation. The experiment shows, that the distributed approach is able to let the transmission rate adaptation of the smart devices converge towards a fair

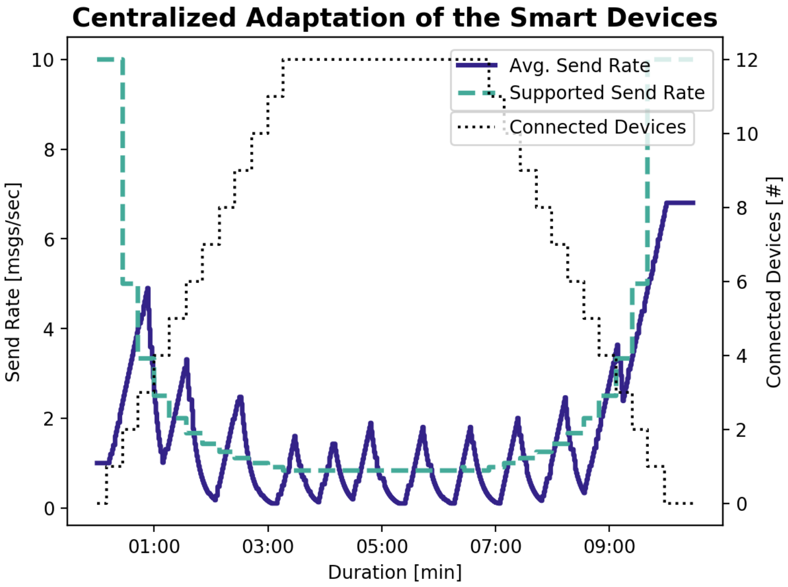


Figure 8.40.: Adaptation behavior and number of the connected devices during the experimental run in a centralized setup.

share of the processing rate of the cloud application. However, compared to the centralized approach, it results in increased amplitudes of the saw-tooth like adaptation behavior, especially for a changing number of connected devices.

### 8.9.4. Discussion

The investigation has shown the applicability of the AIMD scheme in distributed setups for congestion-avoiding cloud applications. However, a distributed setup is associated with a quality reduction of the adaptations. A distributed setup can be considered for smart devices utilizing a wireless sensor link, such that a high frequency is associated with costs. In this case, the available energy level can be used for reconsidering adaptation decisions. As presented in [67], devices can adapt the transmission rate based on changes in the environment, such that the resolution increases if there are many changes in the environment or decrease if there is no change.

# 9. Related Work

This section presents works related to the contributions of the thesis.

## 9.1. Performance Metrics for Scaling Decisions

In this section we discuss works related to auto-scalers and performance metrics. To the best of our knowledge there is no work which evaluates performance metrics for scaling decisions in relation to the microservice characteristics. First, we introduce works related to the evaluation of auto-scalers. Then, we present works related to the evaluation of performance metrics. Finally, we present auto-scalers which are closely related to the context of the evaluation, e.g. by addressing the challenge of scaling microservices with varying resource characteristics.

### 9.1.1. Evaluation of Auto-Scalers

PEAS [56] is an auto-scaling evaluation framework based on the queuing theory. The presented model abstracts cloud applications as a G/G/N stable queue, which capacity is parameterized by a rate based on the average number of requests per time unit that a service instance can handle with an acceptable service time. The proposed model is limited to single service applications and does not allow to optimize auto-scalers considering a complex interaction of services. We have adopted the presented model for the evaluation of performance metrics and refined it to represent single-service Cloud-IoT applications. A queueing theory based model for multi-service applications could be part of future work, e.g. by integrating it with queueing petri nets simulations, as introduced by Kounev [48].

SimuLizar [10] is a performance analysis tool which supports software architects in the design of elastic software systems. However, SimuLizar lacks the capability to model an asynchronous communication via messages queues [47]. Furthermore, it relies on stochastic arrival patterns to induce workload on the system, requiring an extension to support smart device as message inducing and adaptable entities. Therefore, it is in its current state not suitable to evaluate auto-scaling decisions based on message queues.

### 9.1.2. Evaluation of Performance Metrics

Dickel et al. [25] present an evaluation of auto-scaling metrics for scaling stateful IoT gateways. The evaluated metrics are the CPU utilization, the number of concurrent connections and the throughput per gateway. They concluded, that each of the metric on its one is faced with limitations: the CPU utilization does not consider the workload based on the number of concurrent clients and their messaging rate, the number of connected clients does not consider the workload per connection and the throughput per gateway does not consider the number of connected clients. The evaluation lacks elasticity measures to quantify the difference. However, they state, that auto-scaling based on service demands outperforms the CPU utilization. Overall, we deem the work as closely related, since it evaluates performance metrics for scaling decisions. However, whereas it evaluates metrics for auto-scaling, it focuses on CPU-bound (stateful) IoT gateways whereas this evaluation focuses on stateless microservices with varying resource characteristics.

Rao et al. [61] presents a fuzzy control-based resource allocation for cloud applications. They state as one of the challenges the non-uniformity of cloud resources, which means, that users can experience variations in the capacity of the application based on the cloud environment. Based on a E-Commerce benchmark, they measured for a CPU utilization of 80 % variations in the response time up to 150 %. They conclude, that the CPU utilization can not readily translated to application-level performance.

### 9.1.3. Related Auto-Scalers

In this section, we briefly present auto-scalers which overcome the limitations of relying on a specific performance metric, e.g. the CPU limitation.

Control-Theory based works, as presented in [61] or [58] rely on heterogeneous monitoring metrics for scaling decisions since the effectiveness of controllers strong relates to the values of the control parameters. Threshold-based rules approaches, as presented by Bauer et al. [8], address the challenge by relying on a service demand estimation. The service demand denotes the time a unit of work spends on a specific resource, e.g. CPU, and is statistically estimated. They compare the performance to a CPU-based auto-scaler and evaluate, that it achieves a higher degree of elasticity and SLO conformance. The findings support the results, since an auto-scaler based on the service demand is not faced with the same limitations as on the CPU utilization. Khaleg et al. [1] discuss the findings of the performance metric evaluation in this thesis and present an auto-scaling approach which is agnostic towards the microservice characteristics. It consists of a controller, which monitors the QoS of microservices in terms of the response time in order to recalibrate the thresholds through an optimization technique. Based on the characteristics of the microservice they deploy a memory, a CPU and a queue length based auto-scaler.

## 9.2. Feedback Control of Smart Devices

In this section we discuss related work to the flow control of smart devices. First, we discuss feedback control approaches which provide overload protection by congestion control. Then, we present approaches which adjust the collection behavior of smart devices for other non-functional qualities of IoT applications.

### 9.2.1. Congestion Control

A model for congestion control in IoT is presented by Huang et al. [40] with the goal to maintain service access. The model aims to cope with the limited amount of resources in IoT, including network bandwidth and service capacities. It is based on an improved Random Early Discard (RED) algorithm is employed to guarantee a fast response time of the IoT services. The system is modeled as a queue, in which packages are dropped at a certain probability if the queue length exceeds a threshold. Furthermore, the arrival rate will be reduced, which resembles an implicit feedback control. Therefore, the approach aim to maintain the service access by dropping packets. In contrast to the approaches presented in this thesis, it aims to capture bottlenecks including network bandwidth, and does not make any assumption about the services capacity. By using an implicit performance model in our approaches, we aim to fairly share the capacity across the smart devices without dropping messages. Future work may compare the presented approach with ours based on the impact on the data quality.

Nylander et al. proposed control-theoretical approaches for feedback control in [54] and [53]. In [54] they focus on combining the actions of a load balancer with user experience degradation techniques, in order to handle capacity shortages and achieve a predictable response time. In [53] they refine the degradation controller to additionally provide formal guarantees. A load-balancing controller decides based on the response time of services, if the request should be degraded. Whereas we consider this approach as promising to control congestions by data quality reductions we deem the imposed load-balancing architecture as a challenge to its feasibility. The reason is, that we do not consider load-balancing as a concern of the service operator, since it is fully managed by cloud platforms, e.g. CloudFoundry or Kubernetes.

### 9.2.2. Collection Strategies

Siris et al. [67] present approaches to efficiently collect IoT data while achieving a target data accuracy, response time, energy and privacy protection. The context of this work is an IoT platform, which collects data from smart devices in specific intervals. The accuracy-driven approach maintains a target accuracy and adapts the measurement rate based on changes in the data, whereas the time-driven approach adjust the time period between measurement requests in order to ensure, that the elapsed time to the last measurement is below a specific threshold. The energy-driven strategy adjusts the measurement interval with accuracy and energy cost considerations and the privacy-driven strategy adds noise to measurements. The results show, that AIMD adaptations of the measurement periods are robust to different types of measurements. The approaches are closely related to the work of the thesis by utilizing AIMD schemes but differ in goals. In contrast to our work, the proposed time-driven strategy aims to cope with network delays in order to maintain the timeliness of data, whereas our approaches aim to cope with capacity shortages.

Based on the battery limitations of many smart devices energy-efficiency is a concern of collection strategies. Ogawa et al. [55] proposes a measurement rate reduction of field sensing devices in dependency to detected conditions. The approach proposed by Nolan et al. [52] aims to extend the operating lifetime of power-constrained devices by an adaptive messaging rate, to minimize the energy-intensive transmission of data via a wireless link. Whereas the first approach reduces the load on the network and the second approach increases the operating lifetime both do not consider the state of the processing application.

Sato et al. [64] present a feedback control theory based model of real-time IoT applications, which represents smart devices as controlled objects. They introduce a smart station, which notifies a crowd to space evenly by controlling the number of people which have to leave. They analyze the functional performance of the controller using numerical analysis. Whereas the modeling technique is focused on functional capabilities it can be refined to support performance-based goals, e.g. transmission rate adaptations based on the current accuracy.

## 9.3. QoS Optimization of Cloud Applications

Bauer et al. [9] present a systematic search approach for distributed cloud services. By searching for minimal resource configurations in terms of the resource costs for a given load intensity they create a Pareto set. A resource configuration is considered as efficient, if it is able to meet time-based SLOs. However, in order to be applicable to sensing cloud applications, the load intensity should emerge from the number of active smart devices and their transmission rate configuration. Therefore the search problem has to be extended in a dimension with the number of devices in order to obtain the optimal resource configuration in a specific connectivity scenario.

An IoT application model and a back tracking approach is presented by Brogi and Forti [14] to support a QoS-aware deployment of IoT applications on fog infrastructures. First, they introduce QoS profiles, which captures latency and bandwidth features. Then, they model fog infrastructures, consisting of things, nodes, cloud data centers and communication links. Finally, they model the IoT application as a triplet of things, software components and infrastructure. Based on the model they introduce a search which consists of a pre-processing and backtracking search phase to find an eligible deployment. Therefore, the approach focuses on the deployment in the design-phase without considering the elasticity of services or smart device reconfigurations.

Palladio [11] is a method to analyze the QoS properties of component-based applications based on an application model with a focus on performance and reliability. It is extensively used in research and provides a rich tool-support for performance analysts. Klinaku et al. [47] conduct a case study to assess the applicability of Palladio for microservices and cloud technologies. They conclude, that Palladio is able to provide acceptable prediction results for microservices in cloud environments but is in its current state not feasible for elasticity and cost-efficiency analysis.

## 9.4. Frameworks for Self-Adaptive Systems

In the following we present frameworks for self-adaptive systems which are closely related to SEIA.

Garlan et al. [31] present Rainbow, a highly adopted framework for self-adaptive systems, which has been extensively evaluated in terms of effectiveness [20] and robustness [15]. Rainbow can be considered as a reference implementation of the MAPE-K loop and provides functionalities to be extended. However, it aims to support web applications and lacks concepts of cloud applications. We have adopted the graph based representation of applications in our application model. Therefore, we consider the software architecture model and the probe and effector bindings of SEIA as transferable to Rainbow.

HOGNA is a platform for self-adaptive applications in cloud environments presented by Barna et al. [3]. In its core it is a MAPE-K loop, which provides out-of-the-box implementations, e.g. a performance model, tailored to web applications. Besides, it has deployment support based on a topology description. Whereas it aims to provide a platform for the prototype creation and evaluation of adaptive feedback loops, it is tailored to web applications deployed in cloud environments. Therefore it does not provide an application model with effectors and probes to to manage sensing cloud applications.

# 10. Conclusion

In this chapter the thesis is concluded. First, we summarize the contributions of this work in section 10.1. Section 10.2 discusses the benefits of the presented approaches and section 10.3 the assumptions and limitations. In section 10.4 we outline future work.

## 10.1. Summary

This thesis presented QoS cost functions and time-driven flow control approaches to improve the runtime management of sensing cloud applications. The contributions of this thesis address the Research Questions (RQs) discussed in section 1.4. In the following we summarize the contributions:

**C1: Evaluation of the impact of capacity and resource demand variations on scaling decisions.** We obtained insight in selecting performance metrics by evaluating the influence of microservice characteristic variations on the elasticity of threshold-based rules auto-scaler (RQ 1). The elasticity of CPU-based auto-scalers is greatly decreased if the CPU share of processing messages decreases, e.g. due to an increased wait time or computation capacity. Queue-based auto-scalers outperforms CPU-based auto-scalers for scaling microservices with varying wait time. Additionally, the presented model allows to predict the achieved elasticity of an auto-scaler based on the CPU or message queue metrics.

We have shown that thresholds based on message queue metrics are robust towards resource demand and capacity variations. The elasticity of CPU-based auto-scalers is greatly decreased if the CPU share of processing messages decreases, e.g. due to an increased wait time or computation capacity. We have shown, that these result in drastic changes of the SLO conformance for the CPU utilization.

**C2: Design of QoS cost functions for sensing cloud applications and a QoS optimization framework.** The proposed QoS cost functions are able to capture the effect of resource and transmission rate reconfigurations on the qualities of sensing cloud applications (RQ 2). The presented optimization framework allows to integrate a simulation model and optimization method to search for runtime or resource configuration candidates to minimize the cumulative QoS costs.

The presented QoS cost functions aim to be pragmatic, such that a service operator can construct them based on a SLA. A set of cost function can be used to search for an optimal configuration at design time. Based on the availability of most of the presented QoS input metrics in cloud enviroments, e.g. by probing message broker or cloud platforms, the QoS cost functions can also be used for decision making at runtime. Furthemore, we have instantiated the optimization framework using the proposed QoS cost functions to evaluate the presented flow control approaches in case studies.

**C3: Development of flow control approaches to overload protect sensing cloud applications.** The developed flow control approaches address overload situations by adjusting the transmission rate. They aim to maintain the QoS by improving the timeliness on the expense of accuracy. Therefore, they are a complementary runtime mechanism to auto-scaling, which maintains the QoS on the expense of resource costs. The evaluation on the example of a sensing cloud application shows, that the approaches are able to maintain a QoS with varying overload intensities with a comparable performance to auto-scalers (RQ 3).

We presented an approaches based on the congestion avoidance mechanisms of TCP (TIF) and refined them with a capacity-estimation module (CEF). We evaluated that both approaches are able to cope with capacity shortages by transmission rate adjustments. However, they do probe the capacity by inducing minor congestions of an intensity based on the configuration of the approach. We evaluated, that both approaches are efficient enough to perform similar to auto-scalers. Thus they provide a complementary mechanism to auto-scaling to cope with capacity shortages by reducing the load to a degree which avoids congestions.

**C4: Design of rule-based coupling approaches to combine flow control with auto-scaling.** We present rule-based coupling approaches for coupling flow control approaches with auto-scaling. This allows to leverage the mechanisms of auto-scaling and flow control in managing sensing cloud applications. We evaluate, that a coupling is able to improve the QoS conformance (RQ 4).

We proposed rule-set based couplings mechanisms to combine auto-scalers with flow control approaches. Overall, we evaluated three techniques: concurrent, QoS-based rules and fuzzy rules coupling. The efficiency of a concurrent coupling emerges from the coupled strategies, which have to be tuned carefully. In our case study, it has shown to be beneficial to use a rule set to decide which strategy to activate. The QoS-based rules have shown the highest QoS conformance but depend on heavy optimization to properly configure the thresholds. The fuzzy rules are able to stabilize the the QoS conformance in all application scenarios and provide a reusable set of rules. Especially for sensing cloud applications with non-linear cost functions a coupling has resulted in a substantially improved QoS conformance.

**C5: Design of a sensing cloud application model for self-adaptive systems.** We present sensing cloud application model for self-adaptive systems consisting of the cloud topology and effectors and probes. This model enables to design runtime management approaches.

We instantiated the application model with the SEIA framework to conduct experiments on case study systems to empirically answer RQs in multiple publications [32], [33], [34]. Therefore, we demonstrated the adequacy of the abstraction level of the application model by instantiating it for multiple cloud applications architectures in heterogeneous environments (RQ 5). By presenting an application model which contains relevant components of cloud applications, e.g. microservices and message queues, we enable a service operator to design runtime strategies with transmission rate and resource configuration adjustments.

## 10.2. Benefits

The contributions of this thesis enable service operators to improve the QoS conformance of sensing cloud applications by selecting and configuring runtime management approaches based on a set of QoS cost functions. In the following we describe the benefits of the work from the perspective of a service operator.

The QoS cost functions enable a service operator to capture the impact of transmission rate and resource configurations on the QoS of sensing cloud applications. Therefore, it enables to optimize the transmission rate and resource configuration in terms of the cumulative QoS costs in design time. It also enables to create Pareto curves to analyze the trade-offs in resource configuration costs and achievable data quality. Additionally, the configuration of runtime management approaches can be optimized in design time in respect to an application scenario.

The advantages of the flow control approaches are to provide another runtime tool to cope with capacity shortages or to fully utilize the available capacity. They enable a service operator to maintain the QoS in scenarios in which auto-scalers face limitations, e.g. based on economical or resource constraints. Sensing cloud applications may exhibit cost functions which are non-linear, which makes a combination of flow control and auto-scaling beneficial for the QoS conformance, by enabling to cope with capacity shortages with a combination of both. Furthermore, they can be applied to fully utilize the available capacity, e.g. if the cloud application is accuracy-driven.

The presented framework `SEIA` reduces the effort to prototypical develop runtime management approaches and deploy it on heterogeneous environments. With the application model, probes and effectors it enables a tool set to design complex runtime strategies. The framework provides default implementations for adaptation logic, e.g. threshold-based rules auto-scalers, and for cloud environments, e.g. CloudFoundry and Kubernetes, and can be extended by other developers.

## 10.3. Assumptions and Limitations

In the following, we discuss assumptions and limitations of the contributions.

**Performance Metrics Evaluation.** The evaluation of the performance metrics assumed microservices which are stateless. Therefore, the analytical and the simulation model are based on this assumption. This limits the results of the evaluation to scaling stateless microservices only. Furthermore, we assumed microservices which consume messages of a single message queue. Therefore the evaluation of scaling decisions based on queueing metrics is limited to this architectural design. Furthermore, we have abstracted the messages in terms of CPU and wait time. The model and the evaluation assumes, that only homogeneous messages are processed, whereas in practice a microservice may process messages of multiple classes.

**QoS Cost Functions.** The presented set of QoS cost functions assume that the smart devices of a sensing cloud application perceive the environment with one sensor. However, IoT applications often require the input of multiple sensors on the smart devices. Additionally, the QoS cost functions assume a continuous transfer of data, whereas in practice smart devices may transfer data on events. Therefore the QoS cost functions are limited to sensing cloud applications with a single data stream.

**Flow Control.** The presented flow control approaches are based on the assumption, that the load on the cloud application only depends on the transmission configuration of the smart devices. Therefore, they are limited to manage sensing cloud applications with smart devices transmitting only one type of messages. Adaptation decisions are based on the assumption, that each message has an equal resource demand. Whereas the flow control approaches are fair in respect of sharing the capacity equally across all smart device, they may not result in fairness in respect to accuracy. The reason is, that the changes in the environment sensed by each smart device can vary greatly and may require a different sensing rate to achieve the same accuracy. Therefore, the approaches assume that adjusting the transmission rate of each smart devices results in a comparable change of the accuracy. The presented approaches are limited to continuous message transfers and are not suitable to manage event-based message transfers with unpredictable workloads. Furthermore, they heavily rely on message queues to recognize congestions or estimate the capacity of the cloud service. Therefore, they impose the architectural restriction, that the cloud service processes messages by consuming a message queue.

**SEIA.** The presented framework is able to create and operate runtime strategies for sensing cloud applications. A limitation of the framework is the lack of an automated deployment of a cloud topology. In the current state, a service operator has to provide an application model to the framework and bind it to a deployed cloud application, which induce additional effort.

## 10.4. Future Work

In the following we discuss the future work categorized in identified areas.

**Performance Metrics Evaluation.** The performance metrics evaluation is limited to stateless microservices consuming messages of a message queue. Therefore, the evaluation can be extended to case studies with a more complex workflow in order to identify more insight into the metric selection and its impact on the elasticity of auto-scalers.

**Flow Control.** Future flow control approaches could aim to achieve fairness in terms of the accuracy, such that the capacity of a cloud application is shared across the smart devices in a manner, which results in a similar accuracy. Since many IoT applications sense the environment with multiple sensors, it could be beneficial to consider multiple transmission rate configurations per smart device in adaptation decisions. The concern of the presented flow control approaches is to manage the timeliness dimension of sensing cloud applications on the expense of accuracy. Therefore, the approaches can be extended to manage or consider other qualities, e.g. battery or privacy. Furthermore, we have evaluated rules-based coupling mechanisms for flow control approaches with auto-scaling. More sophisticated flow control approaches could be application-aware and make both resource and transmission rate reconfiguration decisions. Future work could also focus on integrate existing approaches of resource demand estimation into the flow control approaches, e.g. LibReDe [68]. This could resolve the architectural constraint of relying on a message queue for performance estimation.

**QoS Cost Functions.** The QoS cost functions only consider data from one sensor. In practice, smart devices sense the environment via multiple sensors. Therefore it is reasonable to introduce an aggregation method in order to support to quantify the overall accuracy of a sensing cloud application based on the accuracy of each sensor.

**SEIA Framework.** Future work can ease the development effort by extending the framework with implementations for probing and effecting existing IoT platforms. Additionally, it can be enriched with additional models, e.g. security or privacy, to increase the capabilities of the framework. Since the application model is graph-based it can be beneficial to enable a transformation of existing application models to `SEIA`, such that an IoT application with runtime approaches can be designed in a tool like SimuLizar and then operated via `SEIA`.

**Validation.** Based on the small case studies systems it can be beneficial to validate the contributions on larger case study systems. This can help to identify the shortcomings of the approaches. Additionally, it would help to identify future research directions based on the experienced challenges.

# Bibliography

[1] A. Abdel Khaleq and I. Ra. “Agnostic Approach for Microservices Autoscaling in Cloud Applications”. In: *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*. 2019, pp. 1411–1415.

[2] Congestion Avoidance. “Van Jacobson,“Congestion Avoidance and Control”, SIGCOMM 1988”. In: ().

[3] Cornel Barna et al. “Hogna: A platform for self-adaptive applications in cloud environments”. In: *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE. 2015, pp. 83–87.

[4] Victor Basili et al. “GQMˆ+ strategies–aligning business strategies with software measurement”. In: *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*. IEEE. 2007, pp. 488–490.

[5] Len Bass, Ingo Weber, and Liming Zhu. *DevOps: A Software Architect’s Perspective*. Addison-Wesley Professional, 2015.

[6] André Bauer, Nikolas Herbst, and Samuel Kounev. “Design and Evaluation of a Proactive, Application-Aware Auto-Scaler: Tutorial Paper”. In: *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*. ACM. 2017, pp. 425–428.

[7] André Bauer et al. “Chameleon: A hybrid, proactive auto-scaling mechanism on a level-playing field”. In: *IEEE Transactions on Parallel and Distributed Systems* 30.4 (2018), pp. 800–813.

[8] André Bauer et al. “On the value of service demand estimation for auto-scaling”. In: *International Conference on Measurement, Modelling and Evaluation of Computing Systems*. Springer. 2018, pp. 142–156.

[9] André Bauer et al. “Systematic Search for Optimal Resource Configurations of Distributed Applications”. In: *Proceedings of Workshop on Evaluations and Measurements in Self-Aware Computing Systems as part of FAS*(IEEE ICAC/SASO) conferences companion*. 2019.

[10] Matthias Becker, Steffen Becker, and Joachim Meyer. “Simulizar: Design-time modeling and performance analysis of self-adaptive systems”. In: *Software Engineering 2013* (2013).

[11] Steffen Becker, Heiko Koziolek, and Ralf Reussner. “Model-Based Performance Prediction with the Palladio Component Model”. In: *Proceedings of the 6th International Workshop on Software and Performance*. WOSP ’07. Buenes Aires, Argentina: Association for Computing Machinery, 2007, pp. 54–65. ISBN: 1595932976. DOI: `10.1145/1216993.1217006`. URL: `https://doi.org/10.1145/1216993.1217006`.

[12] Jacob Benesty et al. "Pearson correlation coefficient". In: *Noise reduction in speech processing*. Springer, 2009, pp. 1–4.

[13] Alessio Botta et al. "Integration of Cloud computing and Internet of Things: A survey". en. In: *Future Generation Computer Systems* 56 (Mar. 2016), pp. 684–700. ISSN: 0167739X. DOI: `10.1016/j.future.2015.09.021`. URL: `http://linkinghub.elsevier.com/retrieve/pii/S0167739X15003015` (visited on 09/14/2017).

[14] Antonio Brogi and Stefano Forti. "QoS-aware deployment of IoT applications through the fog". In: *IEEE Internet of Things Journal* 4 (2017), pp. 1185–1192.

[15] Javier Cámara et al. "Robustness evaluation of the rainbow framework for self-adaptation". In: *Proceedings of the 29th Annual ACM Symposium on Applied Computing*. ACM. 2014, pp. 376–383.

[16] Cyril Cecchinel et al. "An Architecture to Support the Collection of Big Data in the Internet of Things". In: IEEE, June 2014, pp. 442–449. ISBN: 978-1-4799-5069-0. DOI: `10.1109/SERVICES.2014.83`. URL: `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6903302` (visited on 09/14/2017).

[17] Yair Censor. "Pareto optimality in multiobjective problems". In: *Applied Mathematics and Optimization* 4.1 (1977), pp. 41–59.

[18] Inderveer Chana and Sukhpal Singh. "Quality of service and service level agreements for cloud environments: Issues and challenges". In: *Cloud Computing*. Springer, 2014, pp. 51–72.

[19] You Chen et al. "Survey and taxonomy of feature selection algorithms in intrusion detection system". In: *Information security and cryptology*. Springer. 2006, pp. 153–167.

[20] Shang-Wen Cheng, David Garlan, and Bradley Schmerl. "Evaluating the effectiveness of the rainbow self-adaptive system". In: *2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*. IEEE. 2009, pp. 132–141.

[21] Trieu C Chieu et al. "Dynamic scaling of web applications in a virtualized cloud computing environment". In: *2009 IEEE International Conference on e-Business Engineering*. IEEE. 2009, pp. 281–286.

[22] Dah-Ming Chiu and Raj Jain. "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks". In: *Computer Networks and ISDN systems* 17.1 (1989), pp. 1–14.

[23] S. Das and P. N. Suganthan. "Differential Evolution: A Survey of the State-of-the-Art". In: *IEEE Transactions on Evolutionary Computation* 15.1 (2011), pp. 4–31.

[24] Jiang Dejun, Guillaume Pierre, and Chi-Hung Chi. "EC2 performance analysis for resource provisioning of service-oriented applications". In: *Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops*. Springer. 2009, pp. 197–207.

[25] Helge Dickel, Vladimir Podolskiy, and Michael Gerndt. "Evaluation of Autoscaling Metrics for (stateful) IoT Gateways". In: *2019 IEEE 12th Conference on Service-Oriented Computing and Applications (SOCA)*. IEEE. 2019, pp. 17–24.

[26] Nicola Dragoni et al. “Microservices: yesterday, today, and tomorrow”. In: *arXiv preprint arXiv:1606.04036* (2016).

[27] Diego Dujovne et al. “6TiSCH: deterministic IP-enabled industrial internet (of things)”. In: *IEEE Communications Magazine* 52.12 (2014), pp. 36–41.

[28] Kyle Ebersold and Richard Glass. “THE IMPACT OF DISRUPTIVE TECHNOLOGY: THE INTERNET OF THINGS.” In: *Issues in Information Systems* 16.4 (2015).

[29] Ahmed El Rheddane et al. “Elastic message queues”. In: *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*. IEEE. 2014, pp. 17–23.

[30] Martin Fowler and James Lewis. “Microservices”. In: *ThoughtWorks.* (2014). `http://martinfowler.com/articles/microservices.html`. (Visited on 02/17/2015).

[31] David Garlan et al. “Rainbow: Architecture-based self-adaptation with reusable infrastructure”. In: *Computer* 37.10 (2004), pp. 46–54.

[32] Manuel Gotin, Felix Lösch, and Ralf Reussner. “TCP-Inspired Congestion Avoidance for Cloud-IoT Applications”. In: *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*. IEEE. 2019, pp. 5–10.

[33] Manuel Gotin et al. “Investigating Performance Metrics for Scaling Microservices in CloudIoT-Environments”. In: *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering*. ACM. 2018.

[34] Manuel Gotin et al. “Overload Protection of Cloud-IoT Applications by Feedback Control of Smart Devices”. In: *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*. ACM. 2019, pp. 51–58.

[35] Jasmin Guth et al. “A Detailed Analysis of IoT Platform Architectures: Concepts, Similarities, and Differences”. In: *Internet of Everything*. Springer, 2018, pp. 81–101.

[36] R Hassan et al. “Adaptive congestion control mechanism in CoAP Application Protocol for Internet of Things (IoT)”. In: (2016), pp. 121–125.

[37] Nikolas Herbst et al. “Quantifying cloud performance and dependability: Taxonomy, metric design, and emerging challenges”. In: *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (ToMPECS)* 3.4 (2018), p. 19.

[38] Nikolas Herbst et al. “Ready for Rain? A View from SPEC Research on the Future of Cloud Metrics”. In: *arXiv preprint arXiv:1604.03470* (2016).

[39] Nikolas Roman Herbst, Samuel Kounev, and Ralf Reussner. “Elasticity in cloud computing: What it is, and what it is not”. In: *Proceedings of the 10th International Conference on Autonomic Computing ({ICAC} 13)*. 2013, pp. 23–27.

[40] Jun Huang et al. “Modeling and analysis on congestion control in the Internet of Things”. In: *Communications (ICC), 2014 IEEE International Conference on*. IEEE. 2014, pp. 434–439.

[41] IBM. “An architectural blueprint for autonomic computing”. In: *IBM White Paper* 36.June (2006), p. 34. ISSN: 19448244. DOI: `10.1021/am900608j`. URL: `http://users.encs.concordia.ca/%7B~%7Dac/ac-resources/AC%7B%5C_%7DBlueprint%7B%5C_%7DWhite%7B%5C_%7DPaper%7B%5C_%7D4th.pdf`.

[42] Waheed Iqbal et al. “Adaptive resource provisioning for read intensive multi-tier applications in the cloud”. In: *Future Generation Computer Systems* 27.6 (2011), pp. 871–879.

[43] Rajendra K Jain, Dah-Ming W Chiu, and William R Hawe. “A Quantitative Measure of Fairness and Discrimination”. In: *Eastern Research Laboratory, Digital Equipment Corporation, Hudson, MA* (1984).

[44] Vineet John and Xia Liu. “A Survey of Distributed Message Broker Queues”. In: *arXiv preprint arXiv:1704.00411* (2017).

[45] David G Kendall. “Stochastic processes occurring in the theory of queues and their analysis by the method of the imbedded Markov chain”. In: *The Annals of Mathematical Statistics* (1953), pp. 338–354.

[46] Anja Klein et al. “Representing data quality for streaming and static data”. In: *2007 IEEE 23rd International Conference on Data Engineering Workshop*. IEEE. 2007, pp. 3–10.

[47] Floriment Klinaku, Dominik Bilgery, and Steffen Becker. “The Applicability of Palladio for Assessing the Quality of Cloud-based Microservice Architectures”. In: *Proceedings of the 13th European Conference on Software Architecture - Volume 2*. ECSA ’19. Paris, France: ACM, 2019, pp. 34–37. ISBN: 978-1-4503-7142-1. DOI: 10.1145/3344948.3344961. URL: http://doi.acm.org/10.1145/3344948.3344961.

[48] Samuel Kounev. “Performance modeling and evaluation of distributed component-based systems using queueing petri nets”. In: *IEEE Transactions on Software Engineering* 32.7 (2006), pp. 486–502.

[49] Tania Lorido-Botran, Jose Miguel-Alonso, and Jose A. Lozano. “A Review of Autoscaling Techniques for Elastic Applications in Cloud Environments”. en. In: *Journal of Grid Computing* 12.4 (Dec. 2014), pp. 559–592. ISSN: 1570-7873, 1572-9184. DOI: 10.1007/s10723-014-9314-7. URL: http://link.springer.com/10.1007/s10723-014-9314-7 (visited on 09/14/2017).

[50] M. Marjani et al. “Big IoT Data Analytics: Architecture, Opportunities, and Open Research Challenges”. In: *IEEE Access* 5 (2017), pp. 5247–5261.

[51] Adnan M Mulaosmanović. “Application of the HART protocol for communication with smart field devices”. In: *Vojnotehnički glasnik* 63.3 (2015), pp. 160–175.

[52] Keith E Nolan et al. “Techniques for resilient real-world IoT”. In: *Wireless Communications and Mobile Computing Conference (IWCMC), 2016 International*. IEEE. 2016, pp. 222–226.

[53] Tommi Nylander et al. “BrownoutCC: Cascaded Control for Bounding the Response Times of Cloud Applications”. In: *American Control Conference 2018*. IEEE–Institute of Electrical and Electronics Engineers Inc. 2018.

[54] Tommi Nylander et al. “Cloud Application Predictability through Integrated Load-Balancing and Service Time Control”. In: *2018 IEEE International Conference on Autonomic Computing (ICAC)*. IEEE. 2018, pp. 51–60.

[55] Keigo Ogawa et al. "Edge-centric field monitoring system for energy-efficient and network-friendly field sensing". In: *Consumer Communications & Networking Conference (CCNC), 2018 15th IEEE Annual*. IEEE. 2018, pp. 1–6.

[56] Alessandro Vittorio Papadopoulos et al. "PEAS: A performance evaluation framework for auto-scaling strategies in cloud applications". In: *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)* 1.4 (2016), p. 15.

[57] Ismael Pena-Lopez et al. "ITU Internet report 2005: the internet of things". In: (2005).

[58] Valerio Persico et al. "A Fuzzy Approach Based on Heterogeneous Metrics for Scaling Out Public Clouds". In: *IEEE Transactions on Parallel and Distributed Systems* 28.8 (2017), pp. 2117–2130.

[59] Valerio Persico et al. "Measuring network throughput in the cloud: the case of amazon ec2". In: *Computer Networks* 93 (2015), pp. 408–422.

[60] Valerio Persico et al. "On network throughput variability in microsoft azure cloud". In: *Global Communications Conference (GLOBECOM), 2015 IEEE*. IEEE. 2015, pp. 1–6.

[61] Jia Rao et al. "QoS guarantees and service differentiation for dynamic cloud applications". In: *IEEE Transactions on Network and Service Management* 10.1 (2013), pp. 43–55.

[62] Yousef Rastegari and Fereidoon Shams. "Optimal decomposition of service level objectives into policy assertions". In: *The Scientific World Journal* 2015 (2015).

[63] Per Runeson and Martin Höst. "Guidelines for conducting and reporting case study research in software engineering". In: *Empirical software engineering* 14.2 (2009), p. 131.

[64] Keisuke Sato et al. "A modeling technique utilizing feedback control theory for performance evaluation of IoT system in real-time". In: *2015 International Conference on Wireless Communications & Signal Processing (WCSP)*. IEEE. 2015, pp. 1–5.

[65] M. Sepesy Maučec et al. "Improved Differential Evolution for Large-Scale Black-Box Optimization". In: *IEEE Access* 6 (2018), pp. 29516–29531.

[66] Parminder Singh et al. "Research on Auto-Scaling of Web Applications in Cloud: Survey, Trends and Future Directions". In: *Scalable Computing: Practice and Experience* 20.2 (2019), pp. 399–432.

[67] Vasilios A Siris et al. "Smart application-aware IoT data collection". In: *Journal of Reliable Intelligent Environments* 5.1 (2019), pp. 17–28.

[68] Simon Spinner et al. "LibReDE: a library for resource demand estimation". In: *Proceedings of the 5th ACM/SPEC international conference on Performance engineering*. ACM. 2014, pp. 227–228.

[69] Rainer Storn and Kenneth Price. "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces". In: *Journal of global optimization* 11.4 (1997), pp. 341–359.

[70] Giacomo Tanganelli, Carlo Vallati, and Enzo Mingozzi. “Ensuring Quality of Service in the Internet of Things”. In: (2018), pp. 139–163.

[71] Bhuvan Urgaonkar et al. “Agile dynamic provisioning of multi-tier internet applications”. In: *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 3.1 (2008), p. 1.

[72] Mario Villamizar et al. “Infrastructure Cost Comparison of Running Web Applications in the Cloud Using AWS Lambda and Monolithic and Microservice Architectures”. In: IEEE, May 2016, pp. 179–182. ISBN: 978-1-5090-2453-7. DOI: 10.1109/CCGrid.2016.37. URL: http://ieeexplore.ieee.org/document/7515686/ (visited on 09/14/2017).

[73] Danny Weyns. “Software engineering of self-adaptive systems: an organised tour and future challenges”. In: (2017).

[74] Miaomiao Yu, Chunjie Wu, and Fugee Tsung. “Monitoring the data quality of data streams using a two-step control scheme”. In: *IISE Transactions* (2019), pp. 1–14.

# A. Appendix

## A.1. Publikationsliste

Im Folgenden sind die im Zuge dieser Arbeit veröffentlichten Publikationen aufgeführt:

- Gotin, Manuel, et al. “Investigating performance metrics for scaling microservices in cloudiot-environments.” Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering. ACM, 2018. S. 157-167.
- Gotin, Manuel, et al. "Overload Protection of Cloud-IoT Applications by Feedback Control of Smart Devices." Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering. ACM, 2019.
- Gotin, Manuel, Felix Lösch, and Ralf Reussner. "TCP-Inspired Congestion Avoidance for Cloud-IoT Applications." 2019 IEEE International Conference on Software Architecture Companion (ICSA-C). IEEE, 2019.