FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Optimizing the Placement of Stream Processing Operators in the Fog

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering and Internet Computing

eingereicht von

Thomas Hiessl
Matrikelnummer 1126115

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Assistant Prof. Dr.-Ing. Stefan Schulte
Mitwirkung: Christoph Hochreiner

Wien, 29. Mai 2017

Thomas Hiessl	Stefan Schulte

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Optimizing the Placement of Stream Processing Operators in the Fog

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering and Internet Computing

by

Thomas Hiessl
Registration Number 1126115

to the Faculty of Informatics

at the TU Wien

Advisor: Assistant Prof. Dr.-Ing. Stefan Schulte
Assistance: Christoph Hochreiner

Vienna, 29th May, 2017

Thomas Hiessl	Stefan Schulte

Erklärung zur Verfassung der Arbeit

Thomas Hiessl
Zentagasse 1, 1050 Wien

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

Wien, 29. Mai 2017

Thomas Hiessl

Acknowledgements

Foremost, I would like to thank my supervisors Dr. -Ing. Stefan Schulte and Christoph Hochreiner for their significant supervision and beneficial guidance towards the direction of this thesis. In particular, I owe Christoph Hochreiner my gratitude for his extensive and instant feedback at all times. He encouraged me in terms of scientific methods, technical know-how and inspired me with useful hints for solution approaches. In addition, Matteo Nardelli needs to be mentioned for his provision of experiences and ideas on DSP optimization.

Moreover, my thanks go to my fellow students who have accompanied me along this journey: Kevin Bachmann for sharing his ideas and thoughts on fog computing; Michael Mittermayr for his insights on software technology; Johannes Matt for his scientific support; and Lukas Arneth for being such a great friend.

Personally, I am truly grateful for my parents, Leopoldine and Walter, as well as my family who have shown continuous emotional support during all phases of my life.

Likewise, my girlfriend Victoria deserves many words of thanks for her tremendous patience and understanding during this stressful time. Due to her mental and academic support, she pushed me to finally finish this diploma thesis.

Lastly, I want to show great appreciation for my work colleagues. They were constantly supportive and understanding during my completion of this diploma thesis.

Kurzfassung

Das Internet of Things (IoT) gewinnt mit steigender Zahl an verbundenen Geräten an Bedeutung. Privates und geschäftliches Leben wird durch intelligente Geräte mit Sensoren und Aktuatoren erleichtert. Vorallem in der Fertigungsindustrie sorgt das IoT für verbesserte Prozesse.

Die Sensor-Datenströme werden dazu häufig von Data Stream Processing (DSP)-Topologien in Echtzeit in der Cloud verarbeitet. Da die Sensoren an unterschiedlichsten Standorten platziert sind, kann beim Daten-Upload eine hohe Latenz auftreten. Um diese zu beschränken, werden Fog Computing-Ressourcen verwendet, die ihre Rechenleistung in geografischer Nähe zur Verfügung stellen. Fog Computing ist eine virtualisierte, skalierbare und on-demand zugängliche Rechenplattform, bestehend aus einer großen Anzahl von heterogenen Ressourcen wie z.B. Router, Switches und diverse Endgeräte. Um bestmögliche Servicequalität anbieten zu können, ist es vorteilhaft die DSP-Operatoren auf diversen Fog Resourcen hinsichtlich Latenz, Verfügbarkeit und Kosten zu optimieren.

Das Ziel dieser Arbeit ist die Optimierung von DSP-Operatoren auf Fog Resourcen. Dafür wurde der ODR Reasoner entwickelt, ein loser gekoppelter Service, der für die Optimierung von DSP-Operator-Platzierungen in Fog- und Cloud-Umgebungen eingesetzt wird. Der ODR Reasoner überwacht das Netzwerk- und die DSP-Topologie, um diese Informationen in ein lineares Programm zu integrieren, welches wiederholt gelöst wird. Um zu untersuchen, welche Kriterien bei der Optimierung berücksichtigt werden müssen, untersuchen wir aktuelle Ansätze und analysieren DSP- und Fog-Computing-Eigenschaften. Der entwickelte dynamische Ansatz wird in einer simulierten Fog-Umgebung basierend auf virtuellen Maschinen evaluiert.

Der ODR Reasoner wurde mit einem statischen Ansatz verglichen, der nur beim Start der Topology optimiert. Die Ergebnisse zeigen, dass Latenz und Kosten um 31,5% bzw. 8,8% verringert werden konnten.

Abstract

The increasing presence of connected devices that interact with their environments, is a driving force for the Internet of Things (IoT). In the IoT, various devices are equipped with sensors to support private as well as business life by collecting and providing insightful information. Especially in the manufacturing domain, IoT devices allow for a better organization and reduce the need for human control.

To process the collected IoT data, batch jobs are often executed on large databases that are hosted in the Cloud. Nevertheless, this is not feasible when results are required in real time. Therefore, Data Stream Processing (DSP) considers to process data based on a topology of DSP operators. These DSP operators continuously query data to perform e.g., customized analysis operations. In contrast to batch job processing, DSP operators consider to process a small amount of data only once or a limited number of times in order to deliver results in real time.

IoT data is often streaming from multiple sources to Cloud Resources that enact DSP topologies. This can lead to high latencies when data is uploaded. For that, Fog Computing is used to provide a highly virtualized computing platform between the Cloud and the edge of the network. The Fog consists of a high number of heterogeneous resources to provide scalability and on-demand accessibility. For this, device types such as routers, switches, or even end-user hardware can be used to host e.g., DSP operators. Due to the given heterogeneity of resources, the optimization of DSP operator placements is desirable to achieve a high availability, low latency, and ideally low cost.

The goal of this thesis is to show how the Fog paradigm facilitates DSP systems to improve the Quality of Service. For this, we introduce the ODR Reasoner, a loosely coupled service that is used for the optimization of DSP operator placements in Fog and Cloud environments. The ODR Reasoner continuously monitors the Fog network and DSP topology enactment to incorporate this information into a periodically solved Integer Linear Program optimization model. To investigate which criteria have to be considered in the optimization, we review state-of-the-art approaches and study DSP and Fog Computing characteristics. To evaluate the developed dynamic approach, we test operator placement optimization in a Fog-like test bed.

Our results show that response time and cost is reduced by 31.5% and 8.8% respectively. We compared our ODR approach to a baseline approach that optimizes only when the DSP topology is initially deployed. Furthermore, we learned that achieving a global optimum in operator placement problems is a difficult task for growing Fog networks.

Contents

CHAPTER 1

Introduction

1.1 Motivation

The last years have shown that the Internet of Things (IoT) emerged from its theoretical principles to an important concept for various practical applications. Gubbi et al. [1] refer to sensor networks that produce an enormous amount of data, which has to be stored, processed and presented seamlessly. They identified potential IoT applications for traffic management, infrastructure monitoring, emergency services, healthcare and the manufacturing industry. Especially the latter one is a prominent research area, known as Industry 4.0 in Europe. Increasing manufacturing support and optimization with IoT digitalization concepts are considered as enablers for the next industrial revolution [2].

To cope with upcoming challenges in IoT [3], especially the increase of data that needs to be processed, it is necessary to extend the currently prominent concepts of Cloud Computing by leveraging further computational resources. The Cloud Computing paradigm provides flexible and scalable computational resources, but there is still more potential computing power at the edge (e.g., smartphones or manufacturing machines) and in intermediary nodes (e.g., routers or switches) of the network [4]. Considering this, Bonomi et al. [3] introduce the Fog Computing paradigm as an extension of the Cloud Computing paradigm. The Fog is described as a virtualized platform for compute, storage and networking services at the edge of the network.

In order to process and analyze data (e.g., originating from IoT devices) on Fog and Cloud computing infrastructures, we consider Data Stream Processing (DSP) topologies which are choreographies of stream processing elements and data sources [5, 6]. In the literature, processing elements refer to data stream operators, which usually carry out well-defined operations (e.g., aggregation, filtering, splits) on received data stream tuples [6]. Furthermore, topologies provide mechanisms to process the data streams within a network produced by different data sources (e.g., sensing devices, social networks,

mobile apps). The resulting analytic outcomes can be used to gain valuable insights [7]. To categorize this approach, Buyya et al. [8] distinguish between *Little Data* (or *Big Stream*) and *Big Data*, where the former considers data that is captured from smart IoT devices, while the latter refers to high volume persistent data in cloud storages. Considering this, DSP topologies are part of the *Little Data* concept, which we will now reflect within the context of an IoT manufacturing use case as a motivational scenario.

1.1.1 IoT Manufacturing Scenario

We consider a European manufacturer with two factories in two countries as depicted in Figure 1.1. The larger factory (Smart Factory 1) has two machines with sensors attached, while solely one machine is placed in Smart Factory 2. Furthermore, the factories each host a cloud computing infrastructure (private Cloud for the whole company, Cloudlet in Smart Factory 1, and a Cloudlet in Smart Factory 2). Herein, Cloudlets can be considered as resource-rich computers like a *cloud in a box*, which can be used by nearby devices [9]. Additionally, a public Cloud is used for avoiding internal resource bottlenecks. In this scenario, the Cloud infrastructure is used for hosting company-specific services and especially DSP topologies. These topologies receive data from availability, productivity, and temperature sensors in order to derive Key Performance Indicators (KPIs) of the production process that can be shown in visual dashboard services. Furthermore, if any irregularities are detected, maintenance actions are taken to avoid serious production problems. However, the company is always aiming for improvements in their deployment flexibility. The deployment locations of the DSP operators vary according to the current situation in the computation infrastructure (e.g, latency, costs of resources, availability). The latency of DSP topologies, for example, can be much higher if the processing data has to be uploaded to the public Cloud instead of processing it directly on company-owned devices that avoid long-lasting data transmissions over the Internet. Therefore, the company has to ensure Quality of Service (QoS) criteria. Especially, avoiding processing bottlenecks and decreasing the operating cost are major goals. Furthermore, already existing but not fully utilized devices within the network of the company can act as deployment locations for hosting operators. This Fog computing oriented resource provisioning approach can result in an efficient deployment strategy which can pay off considering e.g., cost savings, application performance, customer satisfaction and business process efficiency.

1.1.2 Discussion: IoT in Cloud vs Fog

The computing devices within the company network, mentioned in the motivational scenario, form a Fog environment that extends the used Public Cloud. Bonomi et al. [3] state that applications exist that are suitable for deployment locations in the Cloud and the Fog. Both deployment locations provide advantages, especially in the field of data analytics. While applications deployed in the Fog are suitable for low latency or real time data analysis, the cloud is more powerful for long-running data analysis batch processes. Nevertheless, according to Vaquero et al [4] the cloud lacks privacy when

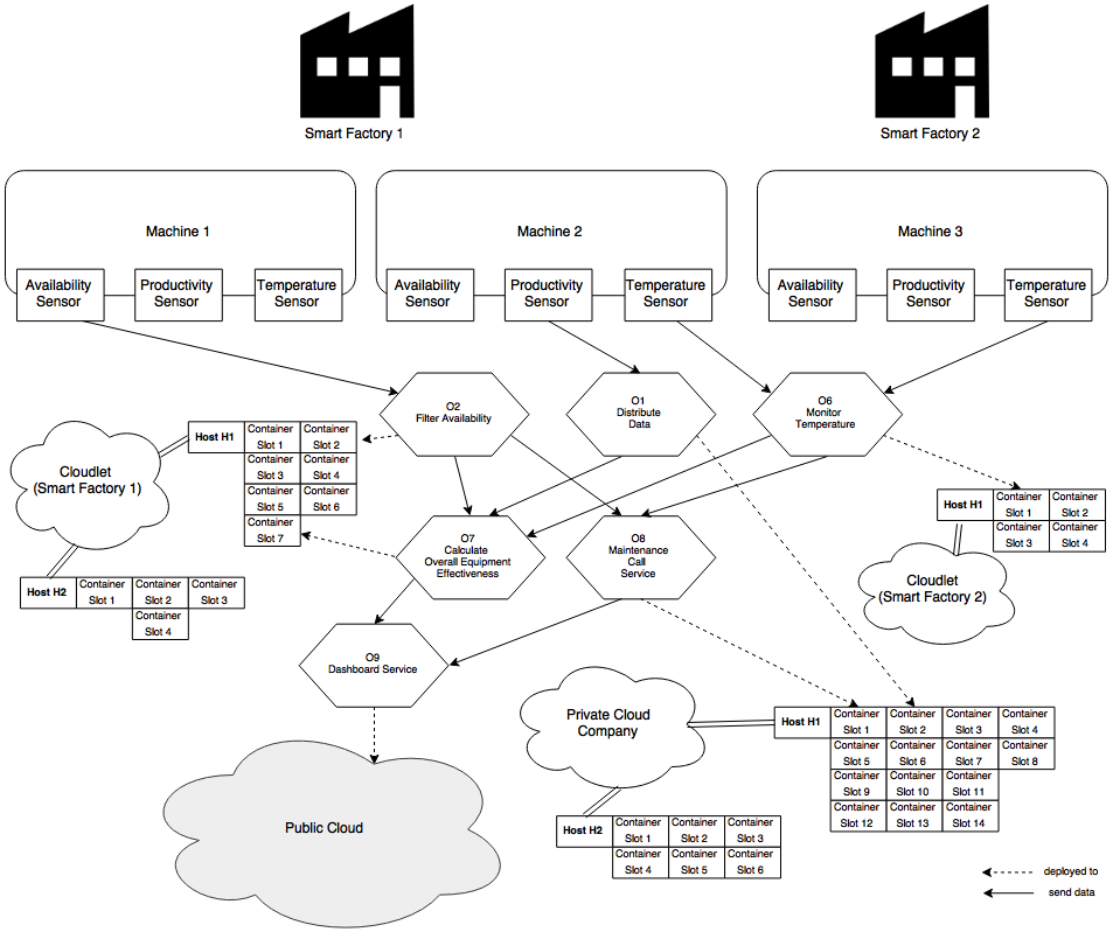


Figure 1.1: Example DSP within Smart Factories

it comes to processing of data that may be restricted to certain persons (data owners) or regions. It has to be ensured that data does not leave a defined network and is processed on local devices. Cardellini et al. [10] discuss deploying DSP operators on different computing nodes in the Fog and Cloud. They optimized operator placements with respect to a heterogeneous infrastructure with different resource capabilities. The authors state that pushing DSP operators or even whole topologies to the cloud could cause excessive stress on the network infrastructure that may lead to network delays. Moving computation to the edge of the network can re-establish the required network performance and improve application scalability. Hochreiner et al. [11] argue that the decomposition of DSP topologies into multiple sets of operators, which can be deployed to a heterogeneous network, is mainly driven by network latency. They point out that the flexibility of having several deployment locations (e.g., Cloud and Fog devices) at different geographic regions can reduce the latency and influence the provisioning cost. For this, the cost can differ significantly between Cloud and Fog Resources [5].

1.2 Problem Statement: Replacement of DSP Topologies

As indicated in Section 1.1.2, a trade-off can exist between latency and cost when it comes to the deployment of services (especially DSP operators) to the Cloud or Fog respectively. Therefore, focusing on DSP topologies for processing IoT data in a continuous way, operator placements need to be optimized to save topology enactment cost and to keep latency at a minimum. Nevertheless, further criteria can be incorporated to achieve a better QoS as considered in existing approaches in the literature. Research in the area of operator placement [10, 12, 13, 14], replication [15] and scaling [16] show that there are multiple ways to achieve optimal operator placement. Most of these approaches consider either a static optimization, where operator placements are computed once or a dynamic distributed optimization, where operator placements are computed continuously from independent decentralized units. Furthermore, heterogeneous Fog networks build a novel environment that has to be regarded when placements are computed, as it was considered by Cardellini et al. [10, 12, 13, 15]. Considering the fog as a flexible network that faces continuous changes of participating devices [9], dynamic replacement can result in better QoS metrics. This requires to periodically monitor the network and use the received metrics to find a new solution for operator placements. Hence, there exist the need to develop an operator placement algorithm that runs for the whole execution time of a DSP topology that has to be optimized. Moreover, the algorithm should find an optimum for the operator placements with respect to the overall changing Fog network. The resulting software implementation contains a mathematical model that is designed in this work. The found operator placement solutions need to be propagated to the DSP topology execution environment to actually update deployments.

1.3 Foundation: VISP and ODP

1.3.1 VISP

Hochreiner et al. [11] proposed the VIenna ecosystem for elastic Stream Processing (VISP). VISP provides a holistic approach for DSP, especially for IoT scenarios. It consists of supporting functionalities along the stream processing lifecycle: Design, Deployment and Execution. Additionally, it also considers the resource and quality elasticity for operator replication. VISP contains two major components namely the VISP Marketplace and VISP Runtime, as depicted in Figure 1.2. The former one is a platform for hosting DSP operator images and designing topologies for DSP scenarios. The latter one is a runtime environment which instantiates, executes, and monitors the created topologies with the respective operators in a distributed way. The Runtime is able to pull operator images from the Marketplace to instantiate new operators. Furthermore, the VISP Runtime has an elasticity component which contains a usage monitor and a reasoner for analyzing whether the system should scale in or out or remain unchanged. Additionally, VISP provides an operator placement framework for a Fog environment.

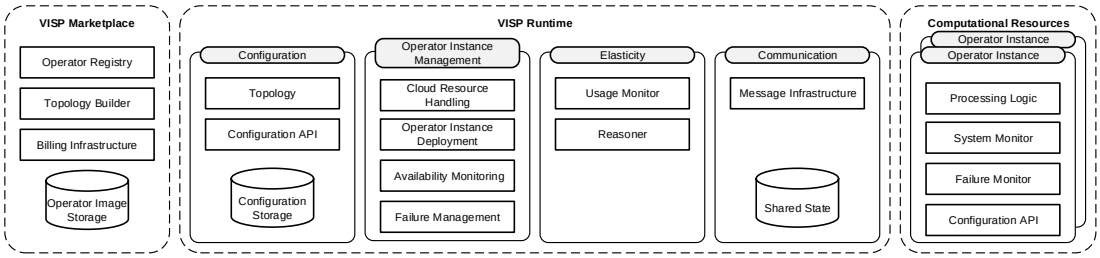


Figure 1.2: VISP Ecosystem [11]

1.3.2 ODP

Cardellini et al. [10] provided an optimization approach formulated as Integer Linear Program (ILP), named Optimal DSP Placement (ODP). The authors focus on a heterogenic Fog network and incorporate its characteristics in the designed ILP model. The implementation of ODP considered *Apache Storm*[1] as a stream processing framework, which accepts initially computed placements of the proposed model and algorithm respectively.

1.4 Aim of the Work

Beside the main foundation presented in Section 1.3, which we use in this work, we further consider research in the area of operator placement [10, 12, 13, 14], replication [15] and scaling [16] from Cardellini et al. as well as elastic data stream processing from Hochreiner et al. [5].

Based on these foundations, the overall goal of this work is to extend VISP [11] with an optimizer component. For this, we present the Optimal DSP Replacement (ODR). The ODP model of Cardellini et al. [10] should be partially considered and extended with expressions and constraints related to additionally identified QoS metrics. Hence, a further goal is to find and incorporate these metrics. Instead of applying an optimization algorithm only at system startup, ODR aims for performing optimized reconfigurations throughout the runtime to cope with ongoing changes on the resource infrastructure and latencies within the Fog network. According to these identified goals, development, and integration issues, the following four Research Questions (RQs) have to be answered:

RQ1 Which criteria are relevant for an operator (re)placement problem?

Since the basic model of the ODP approach [10] takes computation and network latency, as well as availability into account, the VISP-oriented ODP approach needs re-evaluation of metrics that can be provided by VISP and its running operators. Existing approaches in operator placement optimization [10, 12, 13, 14, 17, 18] focus on different metrics that are considered as parameters in different types of algorithms. Therefore, it requires to screen this work to find suitable metrics for the ODR approach.

[1] `http://storm.apache.org/`

RQ2 How can an operator placement problem be defined?

The ODP approach from Cardellini et al. [10] uses an ILP model that defines an objective function and multiple constraints as a mathematical system of equations. Nevertheless, the literature has to be screened for different methods of defining an operator placement problem that can be solved in reasonable time. The gained insights will be used to adapt the ODR model for achieving better operator placements with respect to identified QoS criteria.

RQ3 How can our optimization approach be realized as software and be integrated into established systems?

ODR acts as an extension to a DSP topology execution environment such as the VISP Runtime. Based on the identified criteria that have to be incorporated in the ILP model, it is necessary to identify interfaces which can be used to request the relevant metrics. The communication protocol has to be designed and potential placement updates need to be pushed to VISP Runtime. The actual optimization component of the software that has to be developed needs to compute the parameters and incorporate them into the ILP model. This model has to have a valid representation that can be read by an ILP solver. Before and after the solution is computed, heuristics need to be used to gain results in a reasonable time. Furthermore, the software has to be designed in a way that it can be extended easily with respect to further metrics and additional behavior.

RQ4 How does the optimization compare against baseline approaches?

During the evaluation of the ODR reasoner implementation, the dynamic reconfiguration approach needs to be compared with a baseline. The baseline will be a version of ODR that is only applied once at system startup (static optimization). In the evaluation phase, we will measure the identified QoS metrics and use them for comparison purposes of the two approaches.

1.5 Methodology

To find answers to the identified research questions in Section 1.4 we conduct a literature review and use the acquired knowledge to design the ODR optimization model that is finally implemented in a software prototype. Subsequently, test runs of the software will be performed to produce data that reflects the optimization performance. After evaluating these results, we close with a discussion chapter and outline the future research. The tasks to undertake are presented in the following:

1. **Literature review**
   In order to survey the state of the art of current placement and migration optimization approaches, this work provides a literature review following the approach of Kitchenham et al. [19]. Furthermore, introductory and background topics, such as stream processing and Fog Computing are discussed in this thesis.

2. **Design**
   The design phase requires designing an optimization model that adapts the ILP model of ODP [10]. This model consists of an objective function and constraints. Additionally, further placement problem modeling knowledge - gained from the literature - is regarded to aim for improvements considering the placement within Fog-oriented DSP. Furthermore, the architectural design of the framework has to be created. The integration in the VISP Ecosystem is planned and an adequate binding between the ODR Reasoner and VISP Runtime is established.

3. **Implementation**
   The implementation of the framework, which addresses the challenges discussed in the problem statement, builds on *Java* as a programming language and the *Spring Framework*[2]. For solving the optimization problem, the *IBM CPLEX Optimizer*[3] is used. Finally, the software is integrated with VISP. Furthermore, integration tests are performed in order to ensure correct data exchange between the participating systems.

4. **Evaluation**
   The evaluation considers a sample topology that is used in multiple different test beds. The test beds reflect Cloud and Fog computing infrastructures with their associated characteristics. Operating on these testbeds, the ODR reasoner propagates computed operator placements to VISP such that deployments can be carried out. To measure the success of these replacements the topology execution within the VISP Ecosystem is monitored, the resulting QoS performances are collected, analyzed and interpreted. The tool *tc*[4] is used for network traffic shaping (e.g., slow down the Fog network in certain connections). To produce initial data tuples that get processed by the sample topology, we consider *VISP Dataprovider*[5]. This component ingests tuples into the data source of the DSP topology that is currently executed. It realizes different generation patterns, which help to simulate real world data loads.

1.6 Structure of the Thesis

Based on this introduction chapter, the remainder of the thesis includes the *Background* (see Chapter 2) that covers concepts and paradigms as IoT, Fog Computing, and DSP. These topics lay the foundation of the discussed *Related Work* in Chapter 3. For this, we present and compare DSP frameworks as well as different optimization approaches in this field. In Chapter 4 we describe the *Requirements Analysis & Design* of the ODR Reasoner optimization approach. It includes the definition of the ILP optimization model and presents the software architecture that describes how the placement problem is

[2] https://spring.io/
[3] https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/
[4] https://linux.die.net/man/8/tc
[5] https://github.com/visp-streaming/dataProvider

solved. Furthermore, it shows identified features and tasks that need to be realized in the ODR Reasoner. The *Implementation* chapter (see Chapter 5) provides details of the optimization and the required integration of monitoring data from VISP. Chapter 6 presents the *Evaluation* of this work. Identified Cloud and Fog scenarios as well as the evaluation setup are explained. The corresponding evaluation results are presented and subsequently discussed. To summarize the thesis, in Chapter 7 the *Conclusion* highlights the main results and limitations. Additionally, extension possibilities with respect to ODR are discussed. Future research paths in the field of operator placement optimization in the context of Fog Computing are pointed out.

The referenced URLs in all chapters of this thesis have been accessed on 28$^{\text{th}}$ May, 2017.

CHAPTER 2

Background

This chapter describes important concepts that are used in the presented optimization approach. First, IoT with its enabling technologies and fields of application is discussed. Next, Cloud and Fog Computing are described to understand how resources are provided in these paradigms. Finally, we discuss DSP to conceive this way of real-time data processing, which we want to optimize in this work.

2.1 IoT

2.1.1 Overview

IoT is a concept that considers the presence of connected things or objects that e.g., interact with each other for exchanging information and for cooperation purposes [20]. Connected things like sensing or actuating devices, mobile phones, or Radio-Frequency Identification (RFID) tagged objects aim for reaching a common goal according to a certain field of application [20].

IoT is a result of the convergence of three visions [21]. First, a *Things*-oriented vision considers things with an identifier (e.g., RFID) acting as smart items. Smart items are wireless things that behave autonomously and proactively. They are context-aware and are enabled with suitable sensors and actuators to realize this behavior. The European Commission similarily provides the most recurrent definition [22]:
Things having identities and virtual personalities operating in smart spaces using intelligent interfaces to connect and communicate within social, environmental, and user contexts.

The second vision emphasizes the Internet and the Internet Protocol (IP) as network technology for connecting smart items [21].This *Internet*-oriented vision considers to connect the already existing devices on the Internet (e.g., personal computers, smart phones) with the IoT to have one coherent network. Currently, addressing and reaching

objects with IP is not self-evident for any device within a network as mentioned by Shang et al. [23]. The authors state that IoT networks contain a large number of resource-constrained devices which cannot be always on in order to save energy. These devices mostly require low-energy communication technologies such as IEEE 802.15.4. To enable integration of those devices within an IP network the Internet-oriented vision aims for achieving progress in the IP stack by reducing its complexity [24]. Beside protocol aspects, Buyya et al. [1] consider an architectural perspective for describing the Internet-oriented IoT. An Internet-oriented architecture involves services in the network that consider objects solely as data delivering entities, that are not equipped with functionalities as stated in the Thing-oriented-approach (e.g., context awareness) [1]. According to this, it can be concluded that IoT is an extension of the existing Internet by data transmitting objects called *Things*.

Third, the *Semantic*-oriented vision comprises semantic technologies and reasoning over data that is produced by connected things. Margara et al. [25] mention that in the last few years *stream reasoning* came up that introduces semantics on streaming data. It describes data that is e.g., produced by sensors with annotations in order to specify machine readable details (e.g., measurement precision, data source, spatial and temporal information). One task of stream reasoning is to gain additional insights into streaming data by e.g., acquiring contextual background knowledge [26]. Traffic pattern detection, financial transaction auditing, wind power plant monitoring or monitoring of public-health risks (e.g., epidemics, H1N1 virus spread) are examples for stream reasoning applications [27].

Considering these underlying visions, IoT enables various value-added services, e.g., Atzori et al. [21] state that the introduction of IoT concepts impacts business as well as private fields. Therefore, future developments in fields like healthcare, assisted living, home automation, transportation and logistics, industrial manufacturing etc. will be enabled by IoT [21]. This variety of IoT fields and its estimated potential of making a positive impact on economic development led the US National Intelligence Council (NIC) to declare IoT as a *Disruptive Civil Technology* [28]. In order to conceive this declaration and to reflect the potential of IoT, the ideas for current and future applications are discussed in the remainder of this section.

2.1.2 Fields of Application

Atzori et al. [21] differentiate five application domains and discuss major scenarios. In the following we describe two domains with multiple examples [21, 29, 30]:

1. Transportation and Logistics

   - By equipping vehicles, roads and transported objects with sensors and tags the whole supply chain can be optimized [21]. Sensors send information to

companies that may reroute their vehicles and therefore avoid traffic jams. Furthermore, the timely information of the current state of transported goods and new orders can be used to keep their stock at optimal costs and simultaneously provide the customer with timely delivered products.

- Assisted driving considers driver and passenger support in vehicles [21]. Appropriate information of sensing devices might be used to avoid collisions.
- Environment monitoring (e.g., temperature, humidity) enables food transportation to be executed in a quality-preserving way. For this, sensors are attached to the goods in order to measure environmental conditions. This enables computation of food quality during the whole process within the supply chain, which can result in wasting less food [29].

2. Smart Environments

   - Comfortable homes and offices comprise sensors and actuators that control various processes in houses (e.g., heating, lighting, entertainment systems, intrusion detection) [21].
   - Industrial plants. Connecting machines and automatizing whole production processes is very desirable. Customer order data from enterprise systems can be directly requested from IoT devices attached to manufacturing machines to speed up production and avoid human intervention [21]. The plant manager can track the status of the orders and is aware of the production progress and potential failures within the system. For this, combining IoT devices with existing systems and equipment is necessary. This can be facilitated by a Service Oriented Architecture (SOA)-based approach. This approach considers multiple independent web services that can be orchestrated by a business process execution engine to enact the defined business processes. Adding new IoT devices to control production or to monitor certain manufacturing aspects results in corresponding services that manage access to the device layer. The integration of these services is then considered in the definition of business processes. Finally, their execution uses the IoT device data to e.g., have a global view on factories and production line delays [30].
   - Smart leisure environments. IoT technologies might help to exploit facilities e.g., in gyms better. Training machines can be aware of individual workout plans, which lead to automatic configuration of the training weight as soon as the machine detects the exercising person by reading their RFID tag. The training progress can be tracked and monitoring of health parameters can be used to alarm the exercising person if overtraining is detected [21].

Beside the mentioned applications, Atzori et al. [21] also refer to the domains of *healthcare*, *personal and social*, and *futuristic applications*.

2.1.3 Enabling Technologies

Transforming the IoT visions and concepts into real applications is possible through the consideration of enabling technologies. Atzori et al. [21] differentiate between hardware and software technologies.

Hardware Technologies

Hardware technologies comprise RFID tags and scanners as an identification mechanism of things as well as Wireless Sensor Networks (WSNs) [21]. RFID readers trigger the transmission of IDs from the tag that is attached to certain things. These tags are microchips with an antenna that do not have their own power supply, instead, they harvest the energy from the readers signal. After transmitting the ID to the reader the thing and its application-specific state (e.g., current location within the logistic network) can be monitored. WSNs consist of multiple sensors with their own power supply. Typically, these networks implement the physical and the MAC layer of the OSI Model (OSI) [31] and exchange recorded data between each other in a low-power manner. Due to the missing implementation of the network layer, the WSN needs a device acting as gateway to the Internet to forward data to high level services or middleware implementations.

Software Technologies

IoT Middleware An IoT middleware aims for collecting and distributing sensed data from heterogeneous domains over heterogeneous interfaces [32]. It is a software layer between the application layer and low-level technology layers. A middleware provides an abstraction of technical functionalities as well as things and facilitates application development by taking out the non-application-specific work of developers' hands [21]. Typically, IoT middleware like *HYDRA* [33] unifies functional blocks from fields like security&privacy, device discovery, interoperation within the network, and data volume management [32].

Advances in IoT-enabling software technologies refer primarily to middleware implementations for integrating data from things into an application and still providing the possibility to send commands to underlying actuators [21]. Figure 2.1 depicts an architecture for a suitable IoT middleware based on SOA principles. It defines a thing abstraction layer that handles the thing specific communication and provides a standard web interface for calling exposed methods to read from things as well as to control things. The service management component provides status monitoring, dynamic thing discovery, service configuration and QoS management. As shown in Figure 2.1 each service management instance manages multiple thing abstractions, whereas each thing has a one to one relationship to its abstraction. However, at the top level of the middleware, the service composition combines multiple service management instances into processes. Each process can be triggered by the application that leads to the invocation of exposed service methods on the layer below. After execution, the result is returned to the application

layer. Apart from the vertical invocation of the SOA-based architecture, cross-functional management aspects are considered such as trust, privacy, and security.

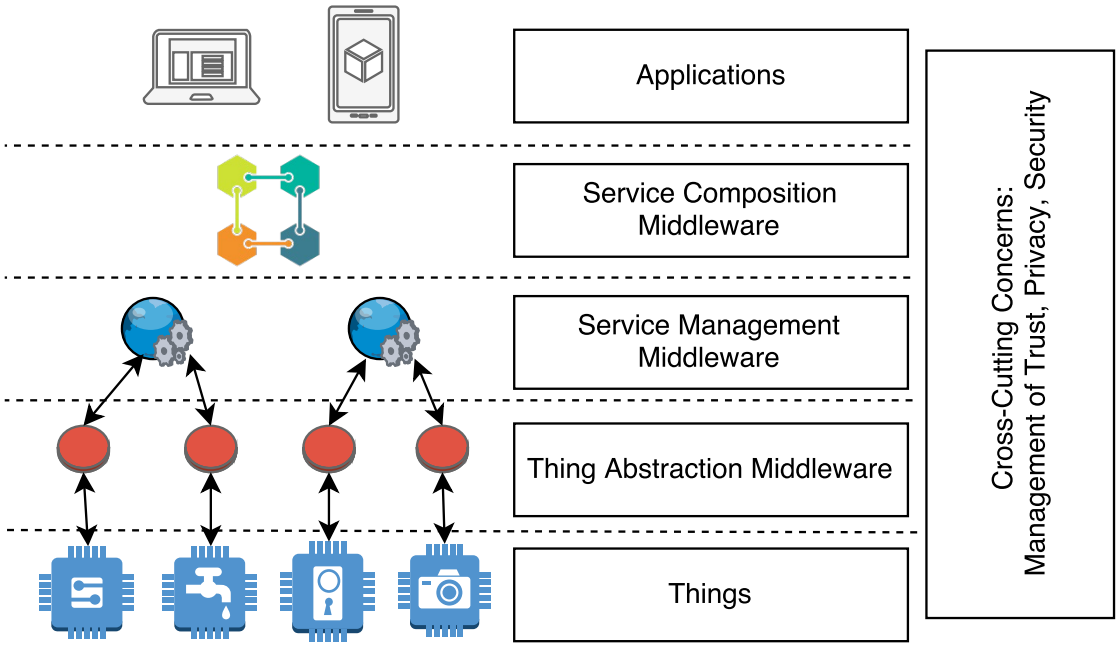


Figure 2.1: SOA-based Architecture for the IoT Middleware [21]

Data Analysis and Presentation Technologies Buyya et al. [1] mention further technologies and important elements that enable the IoT concept and applications. An unprecedented amount of data is created which needs to be stored and analyzed. Therefore, large data centers in the cloud can be used for applying big data technologies for getting insights into the vast amount of structured and unstructured data. Machine learning methods [1] (e.g., neural networks, evolutionary algorithms, support vector machines) are used for achieving automated decision making. Those algorithms scan received data from IoT devices to learn models for prediction purposes (e.g., arrival time of delivered goods based on the current traffic on the road [1, 34]). Furthermore, suitable visualization and interaction mechanisms have been created for touchscreens on multiple smart devices (e.g., tablet, smartphone) [1]. This enables analysis results to be depicted appropriately with possibilities of further extracting valuable information (e.g., drill-down charts).

Application Layer Protocols IoT data streams are processed by using different application layer protocols that differ in their communication style and used low-level protocols [35]. Al-Fuqaha et al. [35] explicitly mention application layer protocols like Constrained Application Protocol (CoAP) and Message Queue Telemetry Transport (MQTT) as enabling technologies for resource constrained IoT devices. CoAP is based on HTTP commands that are exchanged in a request/response manner. Due to the

resource intensive behavior of the TCP protocol, CoAP uses UDP instead. Therefore, it is kept more lightweight because not all requests need to be acknowledged as this would be the case with TCP. In contrast to CoAP, MQTT is based on TCP but prefers a publish/subscribe communication style that results in less data transmission over the network, since data does not need to be requested.

2.1.4 The Role of IoT in Industry

As discussed in Section 2.1.2, the industrial applications of IoT are part of smart environments [21]. The idea of creating such a smart industrial environment made the German government set up a high-tech strategy for modernization. An industrial platform consisting of several industry associations was created and named *Industry 4.0* [2]. Furthermore, this term comprises all activities in this field. According to Lasi et al. [2] this term also describes a future project that is characterized by two development directions. First, *application pull* characterizes the demand for short development cycles, individualized products, flexible production, and resource efficiency. These are the corner stones of Industry 4.0. Second, *technology push* describes technologies for achieving the declared goals. One of that technologies refers to IoT as a concept of digitalization and networking in manufacturing. Especially, machines and produced goods get equipped with additional hardware (see Section 2.1.3). Furthermore, they get connected with other machines and services to be able to act intelligently by perceiving information from their environment. This is achieved by e.g., attaching different sensors and actuators as well as a network-ready devices, which act as a gateway to a suitable IoT middleware and further high-level services to use the collected data in defined business processes (see Section 2.1.3). Considering multiple connected machines in *Smart Factories*, human control will be reduced and the introduced IoT concepts facilitate self-organization within the manufacturing site [36]. Each machine fulfils its tasks by communicating with different entities in the factory or even in the whole supply chain. Ideally, customers only order their individual customized product [36]. Depending on the current stock, machines request the needed resources from their supplier before starting the production process. Change requests of the customer can be regarded at real-time without any human intervention [2]. Kopetz et al. [37] state that IoT can play a major role in reducing maintenance costs of machines. Considering that machines are able to detect anomalies of its components or within the whole factory, they start an automatic maintenance process or call for human intervention. Beside spontaneous incidents, machines might also predict the next maintenance date based on collected data from the past.

2.2 Fog Computing

2.2.1 Overview

According to Bonomi et al. [3], Fog Computing is a paradigm that extends Cloud Computing with additional resources on the edge of the network. As it is known from Cloud Computing [38], several scalable computing resources (e.g., server, storage,

services, applications) are provided for a ubiquitous, convenient, and on-demand access. Nevertheless, these resources are mostly located in dedicated datacenters. Fog Computing further takes into account a large number of heterogenous computing nodes as e.g., smart devices, routers, switches and servers [4]. The following section provides a short recap on the concept of Cloud Computing to better conceive the derived Fog Computing paradigm.

2.2.2 Cloud Computing

Definition

The National Institute of Standards and Technology (NIST) provides a definition [38] of Cloud Computing. The authors describe the paradigm by identifying essential characteristics, service models, and deployment models.

Characteristics The characteristics comprise on-demand self-service, broad network access, resource pooling, rapid elasticity, and automatic monitoring. Cloud consumers therefore have the possibility to access required resources on-demand. Broad network access ensures that connecting to resources can be performed with standard mechanisms from different devices (e.g, HTTP access). Typically, resource pooling enables to share the resources for multiple consumers, which implies significant and efficient reuse and cost savings. The elastic behavior of a Cloud Computing system enables automatic inward or outward scaling of the resources according to the current demand. Herein, we consider inward (outward) scaling as an action for decreasing (increasing) provided resources.

Service Models NIST [38] defines three service models that are used for providing the *Cloud* to consumers. First, Software as a Service (SaaS) considers software applications that run on Cloud infrastructure. Those applications can solely be used, whereas the infrastructure below is not accessible for the consumer. The term *Cloud infrastructure* is defined as the hardware and required software which hosts the application. This infrastructure includes a software abstraction layer that ensures compliance with the mentioned characteristics of the Cloud. This abstraction layer is essential to SaaS since users and application developers would not consider technical details of the underlying infrastructure (e.g., actual storage location of processed SaaS data [39]). The second service model Platform as a Service (PaaS) allows the consumer to create and deploy applications. This model enables control over deployed applications but restricts access to the underlying infrastructure (e.g, network, OS, storage). The PaaS provider may provide a runtime environment for the deployed applications (e.g., Java Virtual Machine (JVM), .NET Runtime) and ready-to-run services like databases, libraries and platform-specific Application Programming Interfaces (APIs) [40]. In the third service model Infrastructure as a Service (IaaS) processing, network, and storage resources that are used to run arbitrary software are provided. These resources can be used via the operating system which the user has control of. Nevertheless, the underlying cloud infrastructure is managed by the IaaS provider.

Deployment Models Finally, NIST considers deployment models that describe the scope of cloud accessibility. Mell et al. [38] distinguish between *Public Cloud*, *Private Cloud*, and *Hybrid Cloud*. The Public Cloud does not restrict the open use, rather it is available to the general public. By contrast, the Private Cloud provides resources only to a certain audience (e.g., employees of a company). The Hybrid Cloud combines Private and Public Clouds in order to use them both for different use cases.

Advances enabled by Cloud Computing

A lot of challenges of the IT industry can be resolved with Cloud Computing [41]. Before the emergence of the Cloud paradigm, IT companies had to buy and run in-house computing equipment, which led to high investment cost and little flexibility. Underprovisioning and overprovisioning happened throughout the operation of the whole IT landscape. Armbrust et al. [41] describe underprovisioning as resource saturation, where all resources are fully utilized. In contrast, overprovisioning is defined as underutilization of used resources. With the advent of Cloud Computing, resources got scalable and the *pay-per-use pricing concept* has been established. Furthermore, the elastic behavior ensures a practical infinite amount of resources that can be used to solve upcoming tasks and to manage peak loads [41]. This feature may also be considered in the Fog Computing paradigm, since Bonomi et al. [3] refer to the Fog as an extension of the Cloud.

2.2.3 Fog Computing Environment

Buyya et al. [7] define Fog Computing as a paradigm that provides cloud-like services on the edge of the network. Cloud and edge services are incorporated into an environment of interacting services and applications, which are hosted on the Cloud or Fog devices respectively. Figure 2.2 depicts these environments of Fog devices and Cloud Computing infrastructure. It shows multiple IoT sensors feeding in data into devices (gateways) close to themselves. These devices and two private *Clouds* are part of the *Fog*. Data is processed and streamed to various devices for further processing, whereas also public *Clouds* are used to exploit resources and to store data. The approach takes advantage of the proximity of sensors to devices which results in low latency [42]. Furthermore, the cloud can be used as a scalable computing model for on-demand peak loads that cannot be handled by Fog devices, since computational-intensive tasks need to be executed as mentioned by Ottenwalder et al. [43].

2.2.4 Fields of Application

Dastjerdi et al. [44] surveyed fields of application of Fog Computing and primarily identified IoT, healthcare, augmented reality, and HTTP web services. The authors mention data trimming as a required and important functionality for IoT devices that continuously receive data from sensors. This can be executed by Fog Resources that

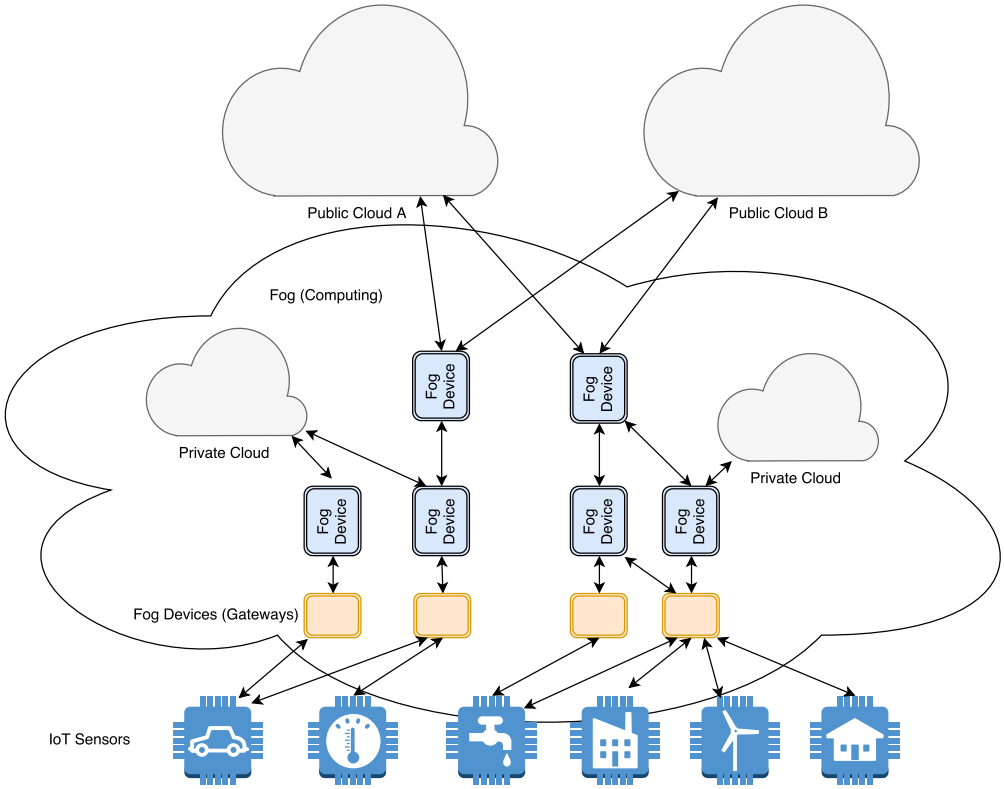


Figure 2.2: Fog Computing Environment [7]

reduce the streaming data to a minimum size so that it can be processed by distant cloud datacenters in a reasonable time.

Fog Computing principles are also used in a fall detection application for stroke patients by Cao et al. [45]. In order to check whether the patient has fallen, the patient's smart phone distributes computation task to servers in the close proximity. These Fog Resources process the tasks in real time and classify the current movements.

Zao et al. [46] proposed an augmented brain computer interface that tracks the brain states of the user in real time by making use of an EEG headset, smartphones and computing services. The EEG signals are streamed to the computing services that are in charge of classifying the state of brain. Considering this information the user is able to play a game which can be controlled just with brain activity. The offloading of complex classification tasks to Fog services is necessary to exploit computational power and to keep the latency low at the same time.

Fog devices can be used to optimize the performance of web browsing as proposed in [47]. These devices have cache functionalities and in case of a network congestion low-resolution images are delivered to the client without requesting the full version from the web server or the Content Delivery Network (CDN).

Dustdar et al. [48] mention that finding a missing child in public areas in cities is a field of application that is currently not leveraged because uploading camera data to the cloud is a privacy concern. Therefore, nearby Fog nodes can fulfill this task by analyzing recorded videos and images.

In the context of the *Industry 4.0*, Gazis et al. [49] see Fog computing as an enabling technology for handling data that is generated by operated machines, vehicles or even whole manufacturing sites. The colossal amount of data that is generated very frequently can not be uploaded to the cloud in a reasonable time to perform suitable analytics (e.g., large refineries may produce 5 TB/day). Therefore, filtering, normalizing and detecting relevant data should be executed in the Fog to avoid heavy network load.

2.2.5 Characteristics

Bonomi et al. [3] state that Fog Computing enables a number of critical IoT services (e.g, Smart cities, connected vehicles) to be executed in the *Fog*. It is seen as a highly virtualized platform that provides computing resources between Cloud Computing datacenters and devices on the edge of the network. To conceive the term of Fog Computing, the authors provide eight defining characteristics:

1. **Low latency, location awareness and mobility**
   Due to execution of tasks on devices that are close to the request issuer, the latency is kept low [3]. According to Stojmenovic et al. [42] the cloud can hardly satisfy location awareness due to the missing mobility of cloud datacenters. On the contrary, in Fog Computing devices can move between locations (e.g., smart phones) and provide specific location-based services, that cannot be offered by the cloud [3]. Furthermore, devices in the Fog take advantage of being in the proximity of potential data sources to improve QoS for network traffic intensive applications. [42].

2. **Wide-spread geographical distribution**
   The Fog considers devices from potentially various geographical regions [3]. Saharan et al. [50] state that the Fog network increases the geographical density of provided resources. This enables that IoT data can be stored in certain regions, which is an important security concern.

3. **Very large number of nodes**
   Multiple resources nodes can participate in the Fog Computing infrastructure [3]. Compared to Cloud Computing the number of server nodes (e.g., Fog devices) can be very high [50], since the number of devices that may provide services increase steadily [4].

4. **Predominant role of wireless access**
   Vaquero et al. [4] state that major improvements in wireless technology (e.g., LTE) make it possible to connect a multitude of devices to the Internet, whereas Cloud Resources are bound to the network of centralized datacenters.

5. **Strong presence of streaming and real-time applications**
   Due to the proximity of some devices to data sources, it is possible to process data in real-time and forward intermediary results to succeeding operators via streams [3, 42].

6. **Heterogeneity**
   The Fog considers different devices as computing nodes [3]. These devices are decentralized and may cooperate to solve tasks. In contrast to Cloud computing, users may not only consume services, they can rather lease part of their devices to provide compute or storage services. Overall, the involved devices form a heterogenous network that may provide incentives for participation [4].

7. **Interoperability**
   The overall execution of joint tasks is performed seamlessly by possibly invoking multiple different resource providers (e.g., stream processing) [3].

8. **On-line analytic**
   IoT Data is received from sources and streamed within the Fog network to provide real time data analysis services [3]. For analytic applications, global centralization of data as well as location information is desired. The former one is provided by a Cloud infrastructure, where all data can be collected in a centralized large storage to apply e.g., historical data analysis on a batch job basis. Location-specific data is provided by decentralized Fog devices that may use these primarily for stream processing purposes [3].

2.2.6 Enabling Technologies and Challenges

Vaquero et al. [4] surveyed existing challenges as well as enabling technologies. The challenges include privacy issues, discovery and synchronization, compute/storage limits, missing standardizations, configuration and operation management for a huge number of devices, accountability, monetisation, and programmability. The accountability and monetisation aspect is crucial for enabling new business models [4]. It needs a system of incentives for sharing Fog Resources between users. Regarding management, the authors state that there will be no full central control over each device in the Fog. Therefore, decentralized and scalable management mechanisms are required. These mechanisms consider proper setup and configuration of participating Fog nodes. Buyya et al. [8] mention that it needs programming models and architectures considering stream and data processing on Fog devices. They highlight that the Fog is a changing environment that faces dynamic adding and removing of Fog Resources. Therefore, an overall management framework is required that needs to consider these characteristics. Considering the security

and privacy aspect, Buyya et al. [8] discuss the need for designing and implementing authentication as well as authorization mechanisms in a network of diverse nodes to establish trust within the Fog. Dustdar et al. [48] refer to naming as a critical challenge that has to be solved for future Fog network realizations, since there is no standardized naming system for Fog purposes which results in a multitude of communication and network protocols that need to be used to integrate heterogeneous Fog devices.

Considering enabling technologies, progress was recorded in the battery lifespan of smart devices (e.g., potential Fog nodes), publish/subscribe protocols for the IoT (e.g., MQTT) [35], the network function virtualization (NFV) [51], software defined networks (SDN) [52], mobile ad-hoc networks [4], and mobile physical connectivity standards like LTE [4]. NFV separates network functions (e.g., router, VPN, firewall) from physical devices and eases dynamic deployment of on-demand network functions to different locations [51]. SDN separates the control logic from underlying network devices (e.g., router) and introduces programmable SDN controllers for the network administration (i.e., the network traffic is controlled by software) [52]. This *softwareisation* [4] with SDN and NFV facilitates the handling of heterogeneous devices in the Fog homogeneously and ideally fully automated by software. It can be concluded that this programmatic network functionalities can support the Fog with e.g., demand-oriented run time adaptations of the network traffic with changes in routing and flexible bandwidth management.

2.3 Data Stream Processing

Data processing is a task that is often executed by storing data in a database and then process it later [6]. However, according to Gama et al. [53] many sources produce data continuously that needs to be processed and analyzed in real time. Furthermore, it is often not possible or acceptable to process a large batch of data from a central storage, because of low latency requirements. Therefore, a paradigm shift from *store and then process* to *on-the-fly processing* models as DSP has been kicked off [6].

Data streams are ordered sequences of data items which are read once or a limited number of times. The processing itself uses bounded computing and storage capabilities [53]. Data streams that are ingested to DSP systems by various sources are characterized as open-ended, changing, and flowing at high-speed. In contrast to traditional Database Management Systems (DBMSs), where humans initiate (predefined) queries usually a few times, DSP systems are querying data throughout time. These queries are applied on streaming data in a certain time window. Typically, a DSP system is expected to produce results in a continuous and timely fashion [54].

2.3.1 Little Data and Big Data

DSP is associated with the term *Little Data* [55]. It refers to transient data collected continuously from humans or physical devices. The more popular term *Big Data* is described as counterpart that comprises persistent knowledge bases spanning multiple

domains to be stored centrally in repositories, Clouds, and large datacenters. Gartner [56] define Big Data as high-volume, high-velocity and high-variety data. High-volume refers to a large amount of data while high-velocity refers to data that is streamed continuously to be processed in real time. The third data property, high-variety, refers to different types of data (e.g., text, video, sensor data).

2.3.2 CEP

A concept which is related to and sometimes combined with DSP is Complex Event Processing (CEP) [57]. The concept of CEP was first explained by Luckham et al. in 1998 [58]. The authors describe CEP as a technology for extracting information from message-based systems. Different granularity of data, regardless of low-level network processing events or high-level enterprise events, can be processed dynamically in order to gain information about systems and its operations. CEP comprises multiple concepts as causal event histories, event patterns, event aggregation and event filtering. Thus, CEP observes and processes correlations between events. Cugola et al. [57] describe DSP systems as generic systems that leave the reasoning over streamed data to their clients. In contrast, CEP systems associate semantics to streamed data items. CEP engines are responsible for processing events and combine them with contextual information to derive higher-level events (often referred to as: *composite events* or *situations*).

2.3.3 Fields of Application

Processing on-line data streams to perform analytics in real-time is relevant for multiple domains as identified by Gedik et al. [6]. The authors refer to live stock and options trading feed in financial services, sensor readings in environmental monitoring and physical link statistics in telecommunications. Goodhope et. al. [59] state that a major trend in modern IT systems is the use of event messages and logs. This information is used in DSP systems for analyzing activities of users and systems that are performed steadily. Typically, in domains as advertising, security, search, relevance determination, and recommendation systems DSP is used for gaining timely and valuable insights into the current behavior of their systems. To mention a concrete example, Goodhope et. al. [59] introduced a *data pipeline* in the social network *LinkedIn*[1]. The authors consider the data pipeline as a real-time feed of messages originated from a publish-subscribe system like *Kafka*[2]. The pipeline handles more than 10 billion messages per day in order to analyze social media aspects.

2.3.4 IoT Data Analytics in the Fog

Low latency and on-line analytics are important points that motivate the use of Fog Resources for IoT scenarios. Stream processing topologies consider a continuous flow

[1] `https://www.linkedin.com/`
[2] `https://kafka.apache.org/`

of data originated from connected *Things* and devices. Data streaming over multiple processing and analysis steps provides users with insightful information [3].

Table 2.1: Data Analytics for IoT Applications in the Fog

Visualization & Reporting (HMI)	
Reporting to Systems & Processes (M2M)	
Collect → Process → Control	Collect → Process (e.g., Filter) → Forward (M2M)

Table 2.1 comprises three layers describing a model for analytics [3]. First, the bottom layer considers different patterns of collecting and processing data from various devices and things. Some analytics scenarios consider a feedback control loop, where actuators start reacting according to the decisions made in the process (analysis) step. Other scenarios take filtering into account before forwarding the data to further devices/machines and to the reporting layer above. This Machine-to-Machine Interaction (M2M) also takes place in the middle layer where application-external systems and processes get informed and updated. Visualization & Reporting is used as a channel of information for Human-to-Machine Interaction (HMI). Stojmenovic et al. [42] state that different types of storages (e.g., ephemeral storage, limited size storage) that are used within the Fog nodes could be a challenge for data analysis purposes such that an integration with existing Cloud storages might be necessary. However, an opportunity of data analytics in the Fog is the potential to achieve real-time analytics and decision making by involving Fog services in the close proximity of data sources [44].

2.3.5 Topologies and Operators

In System S, a large-scale DSP middleware from *IBM*[3], Gedik et al. [6] describe the term *Data-Flow Graphs*, *Processing Elements* (PEs), and *Stream Data Objects* (SDOs). A Data-Flow Graph is a set of multiple PEs, also mentioned as *operators*. Each operator processes data streamed from sources and forwarded to destinations, which are in turn operators. The connection structure between multiple operators and the direction of streamed data is defined in the *Topology* of the data-flow graph. Stream data objects are the data items that are exchanged between the operators. Stream data objects are described as tuples. Typically, operators provide well-defined functions, which can be applied to streamed tuples, e.g., aggregate (grouping), join (correlating multiple input streams), sort, split (routing tuples to different output streams), barrier (synchronization point), functor (tuple-level manipulations such as filtering, projection, mapping, transformation), arbitrary user-defined operations.

[3] https://www.ibm.com/us-en/

In practice, DSP systems like *Apache Storm*[4] use these fundamental concepts of topologies, operators and their functions. Furthermore, operator functions can be added or adapted like it is possible in VISP [11]. Various analytical functions as well as machine learning tasks can be executed by adding customized or specially developed operators (e.g., deployment of operators in *Docker Container*[5]).

[4] http://storm.apache.org/
[5] https://www.docker.com/

CHAPTER 3

Related Work

The main contribution of this work is an optimization of operator placements based on periodic reconfigurations. Therefore, this section provides related work on operator placement approaches and associated topics. First, related work like DSP frameworks and Fog Computing provisioning frameworks are presented. Second, optimization approaches are presented and compared by applying the approach by Kitchenham et al. [19]. Third, related work in other areas, such as service instance scheduling and cloud deployment, provide a broader picture on QoS-based optimizations. Finally, state migration and its relation to placement problems is discussed.

3.1 Data Stream Processing Frameworks

3.1.1 Apache Storm

Apache Storm is a distributed real-time data stream processing system that is able to handle large volumes of high-velocity data. The system is a cluster consisting of three types of nodes. First, *Nimbus* distributes code to the cluster for being executed. This code reflects the behavior that is executed on the operators of the stream processing topology (e.g., filter data tuples). For this, Nimbus considers so-called *worker nodes* as operators, which are monitored continuously. Second, the *Apache ZooKeeper*[1] nodes coordinate the initialized storm cluster by providing services for naming, synchronization, and sharing information between groups. Third, supervisor nodes start and stop worker nodes and maintain a communication channel with the *ZooKeeper*. The Apache Storm DSP topologies consist of different forms of operators like spouts (data sources) and bolts (typical processing elements). Those operators run as part of the cluster to be executed in a scalable, fault-tolerant and reliable way. Scalability is ensured by a parallel execution of the topology, while automatic restarting of failed nodes accounts for fault-tolerance.

[1] `https://zookeeper.apache.org/`

Furthermore, storm guarantees that each tuple is processed exactly once. In case of intermediate failures, messages are replayed to re-establish consistency.

Related Approaches

Twitter Heron[2] is a realtime, distributed, fault-tolerant stream processing engine, which is fully compatible with Apache Storm topologies. The additional features refer to isolation, resource constraints and performance issues. Isolation means that Heron operators are process-based rather than thread-based in order to provide better profiling and troubleshooting possibilities. Resource constraints ensure that no worker node can use more resources than initially allocated, which can happen in Apache Storm topologies. Furthermore, design choices of Heron focused more on performance issues such as throughput and latency.

Beside Apache Storm and Twitter Heron, a further stream processing framework is *Apache Flink*[3] that aims for high-availability. A more general large-scale data processing tool is *Apache Spark*[4] that considers, apart from stream processing, processing of large data batches.

3.1.2 SPADE: The System S Declarative Stream Processing Engine

System S [6] is a large-scale distributed DSP middleware that considers topologies and operators to meet the *on-the-fly processing* paradigm requirements as discussed in Section 2.3.5. As depicted in Figure 3.1, the stream processing core comprises a dataflow graph manager to define the stream connections between the operators. Data transport is handled by the data fabric component, whereas the resource manager component is responsible for placing the operators (called Processing Element (PE) in System S) to hosts that are continuously monitored. During the uptime, the operators are packed in Processing Element Container (PEC) to be provided with a run time context and a security barrier. Stream Processing Application Declarative Engine (SPADE) is used by developers to describe the problem of data processing in a declarative way. System S makes use of a code generation framework to translate the SPADE problem description to operator templates, topology specifications, operator binaries and job descriptions. Those parts are then deployed and executed. Additionally, run time services are deployed that ensure reliability, placement optimization, security and fault tolerance.

3.1.3 VISP

Hochreiner et al. [11] created the VISP ecosystem, a holistic approach for elastic DSP in IoT scenarios, as discussed in Section 1.4. After developing operators, designing topologies, and hosting them on the VISP Marketplace, the topologies can be enacted by the VISP Runtime (see Section 1.3). Deployed as Docker containers (to ease deployment

[2] https://twitter.github.io/heron/
[3] https://flink.apache.org/
[4] http://spark.apache.org/

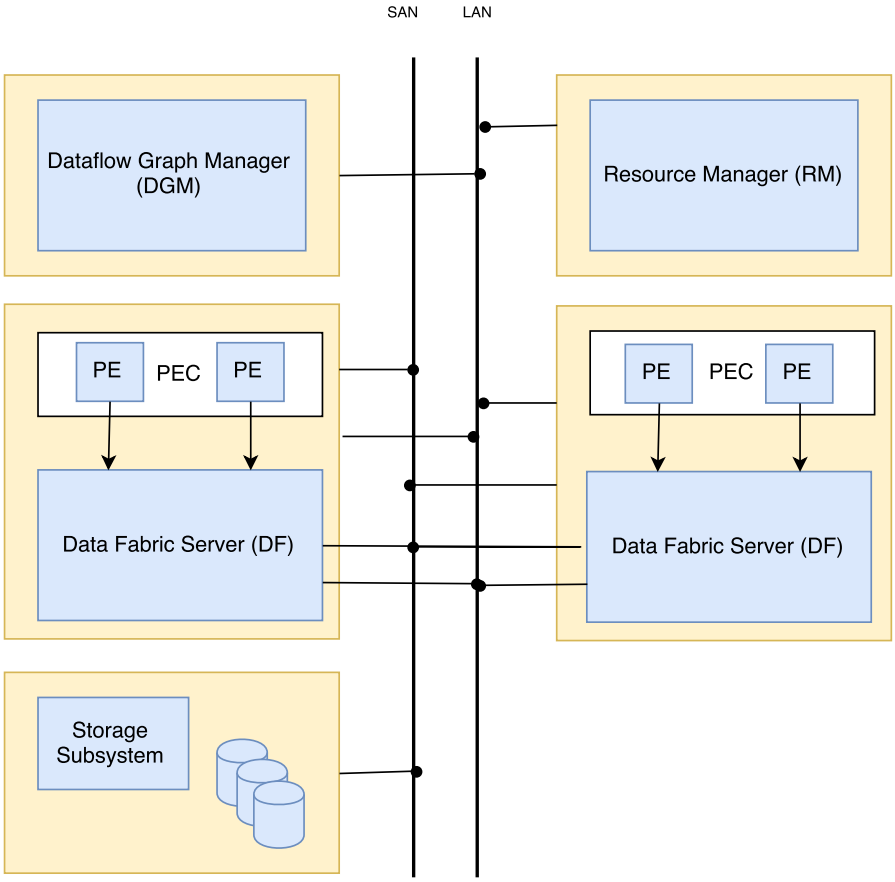


Figure 3.1: Stream Processing Core of System S [60]

for the VISP Runtime), the operators are processing the data streams according to the defined topology on possible heterogeneous resources in the Cloud or Fog. Considered as IoT platform in comparison with prominent frameworks (e.g., *Groovestreams*[5], *Apache Quarks/Edgent*[6], *ThingWorx*[7], *Microsoft Stream Analytics*[8], *Amazon Web Services IoT*[9], *Google Cloud Dataflow*[10]) only VISP provides runtime, marketplace, topology builder, and on-premise deployment together, whereas the other frameworks provide a subset of these functionalities.

[5] https://grovestreams.com/
[6] https://edgent.apache.org/
[7] https://www.thingworx.com/
[8] https://azure.microsoft.com/en-us/services/stream-analytics/
[9] https://aws.amazon.com/de/iot/
[10] https://cloud.google.com/dataflow/

3.1.4 Comparison of DSP Frameworks

We now provide a comparison between the approaches presented in Sections 3.1.1-3.1.3 in tabular form. Therefore, we consider the following identified characteristics:

- **Distributed Runtimes**
  Indicates if the framework considers multiple distributed runtimes for managing and hosting deployed operators.

- **Topology Design Tool**
  Indicates if a tool for creating a topology design consisting of operators and data flows is present.

- **Marketplace**
  Indicates if a marketplace for uploading and pulling new operators for further instantiations is present.

- **Operator replacement**
  Indicates if operators can be replaced at runtime (e.g., according to defined QoS criteria)

- **Language Support**
  Languages that are supported for implementing the operator behavior.

Table 3.1: DSP Frameworks

	Distributed Runtimes	Topology Design Tool	Marketplace	Operator replacement	Language Support
Apache Storm	✓	topologies are defined program-matically	-	out of the box: operators are solely replaced if worker nodes fail [10]	out of the box: Java, Ruby, Python, Javascript, Perl
SPADE System S	✓	✓	-	✓	SPADE Intermediate Language, C++, Java
VISP	✓	✓	✓	✓	not restricted to specific languages

3.2 Fog Computing Provisioning Frameworks

The DSP Framework VISP consider an infrastructure which enables the distribution of operators (packed in a container) to hosts. Multiple VISP Runtimes operate on Cloud and Fog Resources. However, if new devices pop up in the Fog landscape to provide their computing power and storage, VISP requires to manually handle the computing resources in its environment. In general, a Fog Resource management is needed to execute tasks (e.g., VISP Runtime and deployed operators) on a dynamically changing environment. Considering many heterogeneous devices, a provisioning concept is proposed by Skarlat et al. [61]. They present a conceptual framework for Fog Resource provisioning. The provided architecture comprises decentralized Fog cells. A Fog cell is defined as software component that runs on an IoT device and executes services. Multiple Fog cells are connected and build a Fog colony. If some Fog cells need to use additional resources for successful task completion, they request the Fog colony to process necessary (sub)tasks. The required management and orchestration overhead, caused by these task requests, is covered by central Fog orchestration control nodes which in turn are defined as Fog cells.

Bonomi et al. [62] proposed a conceptual Fog computing provisioning framework considering multiple decentralized Fog devices as well as a decentralized database for coordination purposes. Each Fog device hosts a software consisting of two essential components, named *abstraction layer* and *orchestration layer*. The former one accesses the Fog devices' resources (e.g., network, storage, and OS) to host deployed tasks that should be executed within a container. Furthermore, current device statistics like CPU, memory and storage utilization are retrieved from the abstraction layer and sent to the decentralized database. The orchestration layer retrieves policies from the decentralized database to execute them. Typically, these policies specify QoS requirements like minimum delay, assigned CPU power, memory or storage. Additionally, policies for power management are used to account for reasonable energy consumption in the Fog.

3.3 Operator Placement and Related DSP Optimizations

3.3.1 Optimization Possibilities

Several communities use DSP to process data in real-time. DSP has its origin in *Stream Processing* which is a general programming model for efficient and parallel computing [60]. Signal processing, databases, operating systems and complex event processing are examples that make use of *Stream Processing* and related techniques.

Optimization possibilities exist for many stream processing applications, depending on the topology and characteristics of operators. Hirzel et al. [60] propose multiple approaches of core optimizing strategies. First, operator reordering considers two or more filter operators in a sequence, where one of them is more selective in filtering its input stream and therefore sending fewer tuples to the output stream. The concept of selectivity is defined as proportion between input tuples and output tuples. Figure 3.2 depicts how

operators should be moved in case of operator reordering scenarios. In this case, operator B filters out more incoming data tuples than operator A. This leads to a replacement by swapping both operators.

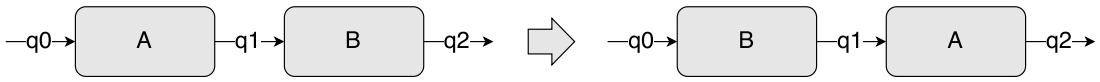


Figure 3.2: Operator Reordering [60]

Second, separating operators into two or multiple sub operators leads to smaller computational steps. Figure 3.3 shows this separation. The two resulting operators can then be used to apply the operator reordering to decrease the latency. Considering DSP on multiple devices, where operators are shared among them, then the computational load can be distributed after the separation was performed.

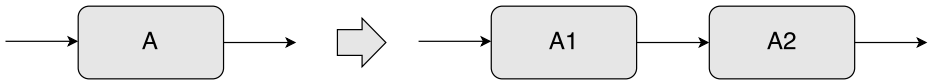


Figure 3.3: Operator Separation [60]

Third, fission is a concept also known as *Data Parallelism* and is shown in Figure 3.4. Seeing operator A as a heavy task that needs a lot of computational power, it makes sense to split it up to multiple equal operators A that process solely a part of the input stream ($1/n$, with n as a number of copies of A). Certainly, data parallelism needs to be applicable to the problem. Furthermore, the computational effort should outweigh the newly introduced network load produced by the copies of A. Multiple cores and/or devices are then involved in solving the overall task A when finally the results are collected in the succeeding merge operator.

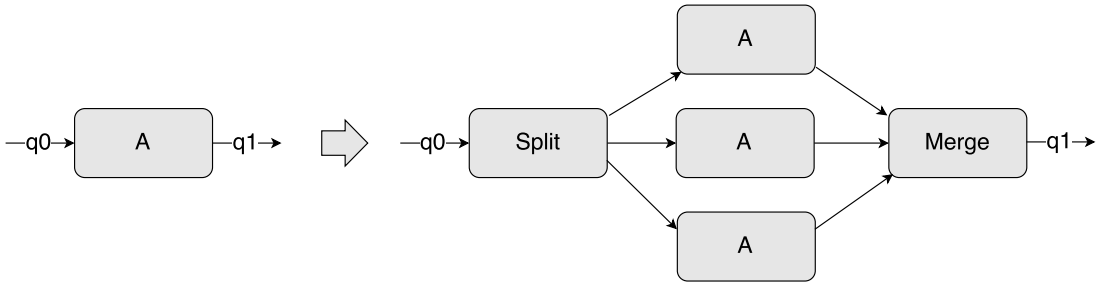


Figure 3.4: Fission (Data Parallelism) [60]

Further concepts of optimizing operators within a topology are described in [60] like redundancy elimination, load balancing, state sharing between operators, batching of streamed tuples, and load shedding.

3.3.2 Operator Placement

The most relevant optimization concept, which is applied in this work, is *Placement* [60] (depicted in Figure 3.5). It considers multiple hosts/devices where operators can be deployed to. Hosts are symbolized as dashed rectangles, spanning multiple operators.

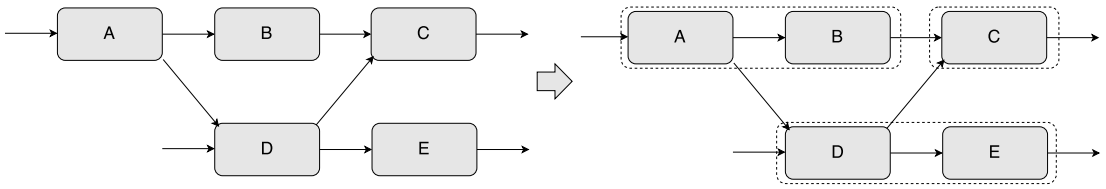


Figure 3.5: Placement: Assign Operators to Hosts [60]

The placement optimization determines the best candidates to host all operators of the topology. Typically, it aims for minimizing performance metrics e.g., latency, network load, and provisioning costs.

Placing operators on different hosts with heterogeneous capacities can be a time consuming and resource intensive task. Especially when considering large topologies and a huge network of hosts (e.g., Fog network), different solution approaches make use of heuristics and local solutions instead of solving the problem with respect to a global optimum. This is necessary to solve the placement problem in reasonable time, since it is NP-hard, as it was shown by Cardellini et al. [10]. The definition of optimality depends on the optimization goals of the approach. Reducing the latency (response time), maximizing the availability, or reducing the costs of stream processing applications are prominent examples for optimizing QoS attributes in DSP systems. These optimizations further ensure scalability of the DSP network as indicated by Rizou et al. [17]. The following sections present and compare various placement approaches, striving for optimal solutions. Furthermore, approaches that apply reconfiguration of placements at runtime or use operator replication and migration techniques are presented as well.

3.3.3 Optimal Operator Placement

In [10], the authors provide a formulation of an optimal DSP placement (ODP) by making use of an Integer Linear Program (ILP) model. The model considers latency, availability, and network load for optimization. Cardellini et al. further take the heterogeneity of DSP applications and the underlying infrastructure into account, by modeling the required and supplied resource capabilities (e.g., CPU). Those parameters as well as the graphs mirroring the resource infrastructure and the topology are incorporated into an objective function and a set of constraints. Heuristics are introduced in the linear program by considering weights for the different QoS criteria (latency, availability, network load) in the objective function. This is done in order to give the user the possibility to adjust the outcome according to the user's preferences. Considering the focus of each part, the user defines weights for selected criteria (e.g., equal focus on response time (latency),

availability, and network loads implies equal weights). The authors evaluate multiple placements by varying the weights to consider also more extreme cases by optimizing solely one QoS attribute.

The placement is executed at startup of the DSP application. A scheduler, which applies the optimization result, places the operators of an Apache Storm topology in a distributed infrastructure. Metrics that need to be considered as parameters are either estimated by a separate component (e.g., by estimating latencies within the network coordinate system [63]) or running a DSP application with any defined assignment to probe the system and harvest the information in such a warm-up phase.

Due to the approach of finding an optimum considering the whole model graphs (DSP topology and resource infrastructure), Cardellini et al. are facing the NP-hardness of ILP solving. The resolution time varies with the size and form of the graphs. Nevertheless, heuristics help to solve it in a feasible amount of time.

3.3.4 Operator Replication and Placement

As an extension to the ODP approach, discussed in the previous section, *Optimal DSP Replication and Placement (ODRP)* is presented in a succeeding publication [15]. ODRP consider two problems for optimization. First, the optimal number of replicas for each operator is determined. This is different in comparison to ODP since ODP considers to deploy only one instance per defined operator. Deploying multiple replicas help to decrease the latency in stream processing applications. Second, the placement is fixed for all replicas. This optimization is carried out without separating it in a two-step approach, rather the number of replicas and its placements are determined within a single-step optimization. Therefore, the ILP model is adapted in comparison to ODP. Herein, a set of multiple hosts is considered as deployment locations for each operator. Beside replication, the ILP model also takes the cost of deployment and data transmission as further optimizable QoS attributes into account. The authors have shown that replication has the potential to keep response time to a minimum when the number of streamed tuples increase. Conversely, overall system cost increase with each replica.

3.3.5 Elastic Stateful Stream Processing

Run-time adaptation of DSP applications are considered in *Elastic Stateful Stream Processing in Storm* [16]. The authors introduce a loosely coupled optimization software that executes scaling actions and performs stateful operator migrations in Apache Storm. Scaling and migration of DSP operators is performed periodically. First, an *Elasticity Manager* checks if the CPU utilization of operators is above or below a defined thresholds. The scale-out action considers a new executor for an operator for each existing operator in overload, while scale-in halves the number of executors when they are not required anymore, i.e., the load drops below a threshold.

After the assignment plan is defined by a *Scheduler* component, a migration notifier triggers operator movements. Newly introduced operators are placed on hosts where

a shared data store among operators within a certain area can be accessed. Due to a limited number of executors that can be placed on worker nodes, relocation is the consequence. The migration solution uses a pause-and-resume approach. It extracts the state that is saved in the shared data store of the previous location, and replays it to a shared data store of the new location when the operator has been migrated. For this, the authors use buffer mechanisms to store the streamed tuples while the operators are moved between locations.

3.3.6 Network-Aware Operator Placement

The authors of *Network-Aware Operator Placement for Stream-Processing Systems* [14] present a decentralized optimization of placements that is performed in a stream-based overlay network (*SBON*). This layer makes placement decisions based on knowledge of stream, network, and node conditions. Without a global knowledge base of the current situation, the network continuously adapts placements according to a multidimensional metric space, called *cost space*. This approach consists of two phases, that are performed decentralized in multiple nodes.

The authors refer to concepts of physics to formulate a model for describing the optimization problem. First, operators are represented as massless bodies connected with other massless bodies or pinned data sources/sinks by a spring. The spring extension and constant are the link latency and data rate respectively. Considering spring relaxation, the massless bodies converge finally to a stable position. This position represents the coordinates of the operators within the cost space, whereas the distance between two operators reflects the cost (network load, caused by latency and data rate) of streaming data over the link. Therefore, the spring relaxation approach leads to a minimum of network load within the system, i.e., $bandwidth * delay^2$ is minimized. Second, resulting coordinates from the previous step (virtual placement) are mapped to the physical space, whereas each operator is assigned to a suitable host. Applying this steps iteratively, the approach regards changes within the infrastructure and the current network situation.

Pietzuch et al. claim that the SBON layer, presented in their solution, can easily be integrated into different distributed DSP systems.

3.3.7 Multi-operator Placement Problem

Rizou et al. present an approach for solving the placement problem with dynamic reconfiguration [17]. Similar, to Pietzuch et al. [14] the authors use a decentralized two-step approach. First, placements are virtually optimized within a latency space by minimizing the network usage of data streams. According to the resulting virtual placement, the virtual node is then mapped to available physical nodes in the second step. In contrast to [14], where $bandwidth * delay^2$ is minimized, Rizou et al. minimizes $bandwidth * delay$.

The presented *Multi Operator Placement (MOP) Algorithm* is following an event-driven manner, whereas operators get informed by neighbor operators in case when their

placements or the data rate of connected links have changed. The minimization problem is solved locally, considering the neighborhood, and the resulting virtual placement within the latency space is mapped to a suitable physical placement. Subsequently, neighborhood operators are informed to adapt. By applying the developed MOP algorithm, at each iteration a local optimum placement is found for operators, whereas in [14] the operators gradually move to a local optimum. To solve the problem for an initial placement, each operator is placed with respect to existing pinned operators. Then the event triggering mechanisms starts to apply a continuous reconfiguration procedure in a decentralized manner.

Rizou et al. state that every operator independently finds its local optimal position by applying the MOP algorithm, which eventually leads to a global optimum.

3.3.8 QoS-aware Scheduling DSP Applications

Further extensions of Apache Storm are presented in [12] and [13]. Self-adaptation capabilities are added to Apache Storm by implementing a new scheduler that dynamically reorganizes the placements of operators. Instead of solving an optimization problem centrally by applying ILP models, the authors present a decentralized approach based on the work of Pietzuch et al. [14]. The approach considers a Fog Computing infrastructure consisting of a large number of heterogeneous devices. The scheduling strategy aims at placing the DSP topology as close as possible to the data sources as well as to the consumer. Multiple nodes keep each other up to date with monitoring data by making use of a gossip-based information dissemination scheme. The *AdaptiveScheduler* is located on each Storm supervisor node that is in charge of monitoring worker nodes. The *AdaptiveScheduler* is therefore responsible for optimization across a group of worker nodes that are capable of executing operators. It executes the distributed scheduling algorithm with respect to a feedback control pattern. This *MAPE* control mechanism is composed of Monitor, Analyze, Plan, and Execute tasks. After the *Monitor* phase, the *Analyze* phase determines if an operator has to be moved to other nodes. During the *Plan* phase candidate nodes (hosts) are identified and in the *Execute* phase the candidates are communicated to the ZooKeeper which performs the new placement. These phases are repeated in a fixed iteration cycle to continuously adapt throughout the runtime.

The authors have shown that latency and inter-node traffic can be reduced by 35% and 31% respectively in comparison to the centralized Apache Storm default scheduler, which does not perform any run-time adaptations. Especially when it comes to changes in resource utilization, the presented approach dynamically replaces the operators, which lead to an even higher advantage in terms of latency compared to the default variant. Nevertheless, during the time of reconfiguration, the DSP topology is in a cool-down state and is not processing any tuples. This leads to a short phase of instability with respect to the defined QoS attributes.

3.3.9 Network-Aware Query Processing

Ahmad et al. present in [18] optimization approaches based on a greedy algorithm. It comprises multiple centralized as well as decentralized algorithms, whereas more advanced approaches are based on the basic ones.

The basic placement approach considers iterating over all operators within the DSP topology in a post-order manner. The candidate set of nodes, where the operators can be placed, is restricted to (1) one of possibly many children locations, (2) a common location with all children, or (3) data sinks. Herein, children are the operators that are successors of a given operator in a given topology. Based on a cost function, the decision for one of these three options is made. Costs are increasing if operators are placed on a different node than its parents or children respectively. Otherwise, when the same nodes are shared, the costs are 0.

The second approach extends the previous one by adding latency considerations to the cost function. Therefore, network distances between the nodes are considered. Placement options with large distances are assigned with higher costs.

The third version further considers a restricted set of candidates, where operators can be placed on. This set is computed so that network distances to the currently used nodes are reduced to a defined limit. Therefore, the algorithm avoids placing operators too far away from its children operators, which could cause high latency.

Due to restriction in scalability and effectiveness of centralized optimizations, all discussed approaches are further provided as decentralized approaches. For this, the node network is separated into multiple zones with one coordinator each. These coordinators are responsible for executing the presented algorithms for a sub tree of the overall DSP topology. Communication between the coordinators is done in order to distribute sub trees that have to be placed, and to exchange information about executed placements of adjacent operators. The distributed protocol makes use of Distributed Hash Table (DHT) primitives to improve scalability, fault tolerance, and look-up efficiency.

3.3.10 Comparison of DSP Optimization Approaches

In the last Sections 3.3.3-3.3.9 we have discussed different optimization approaches for DSP frameworks. To compare these, key characteristics are extracted as follows:

- **Aim of Optimization**
  Characterizes the approach with respect to its optimization goals (e.g., placement, replication)
- **Algorithm**
  Indicates, which kind of algorithm is used to solve the problem.
- **Degree of Decentralization**
  Centralized, partially decentralized, decentralized.

- **Parameters**
  Monitored metrics or initially provided parameters to optimize the respective QoS attribute (e.g., latency, costs).

- **Dynamic Reconfiguration**
  Indicates if the optimization is executed dynamically throughout the run time of the DSP System.

- **Change Trigger**
  Trigger that causes reconfiguration, if implemented.

These characteristics are based on [64]. Table 3.2 uses the identified characteristics to differentiate the discussed optimization approaches.

Table 3.2: DSP Optimization Approaches

	Aim of Optimization	Algorithm	Degree of Decentralizat.	Parameters	Dynamic Reconf.	Change Trigger
ODP [10]	Placement	ILP Optimization	Centralized	Latency, availability, data rate	-	-
ODRP [15]	Placement & Replication	ILP Optimization	Centralized	Latency, availability, data rate, cost	-	-
Elastic Storm [16]	Replication	Threshold-based scaling	Partially decentralized	CPU utilization	✓	Threshold
SBON [14]	Placement	Spring Relaxation (Physics)	Decentralized	Latency, data rate	✓	Periodic
MOP [17]	Placement	Network usage minimization (Gradient Descent)	Decentralized	Latency, data rate	✓	Event-based
QoS-aware Scheduling [12]	Placement	Feedback control loop based on SBON	Partially decentralized	Intra-node(utilization, availability), Inter-node(latency, data rate)	✓	Periodic
Network-Aware Query Processing [18]	Placement	Greedy	Partially decentralized	Latency	✓	Periodic

To conceive the comparison, a few comments follow to clarify some classifications of the approaches. First, *Partially decentralized* refers to the fact that neither a centralized approach, with solely one optimizing component, nor a fully decentralized approach, with one optimization component per operator or working node, is used. In these cases, the optimization is often executed in intermediate nodes as *Coordinators* in [18] that are responsible for defined sub zones of the network. Furthermore, an approach can also be declared as *Partially decentralized* if some tasks are executed centrally, while others are delegated to nodes at the edge of the network, e.g., on Fog nodes. The gradient descent algorithm, used by Rizou et al. in [17], seeks for a minimum of the network usage by incrementally moving along the direction where the defined cost function decreases. Threshold-based change triggers consider a periodic re-evaluation. Nevertheless, actions that change the placements or replicas are only taken if the threshold is exceeded.

For comparison, it first needs a separation between approaches that mainly perform placement optimizations and other DSP optimization approaches. *Elastic Storm* [16] primarily focuses on the extension of *Apache Storm* with elasticity capabilities. The topology execution is monitored and in case of exceeding thresholds, operator replicas are created. These replicas have to be placed but without specific optimization considerations. The main goal is to keep the DSP performance high by unloading work from existing operators, i.e., reducing the amount of data tuples that need to be processed.

In regard to placement optimizations, *ODP* and *ODRP* do not perform reconfigurations of placements throughout the run time of the DSP topology. Their focus is to create a promising initial statement, whereas *ODRP* further instantiates multiple replicas of operators to balance the load. These ILP optimizations consider, beside network-specific metrics, also the availability of nodes and operating costs (at least ODRP). The reconfiguration approaches continuously monitor the current network situation and perform replacements to keep QoS attributes at a certain level.

Due to the NP-hardness of the placement problem [10], heuristics are considered throughout the listed approaches. Furthermore, the strategy of decentralizing the algorithm execution was used to avoid bottlenecks in centralized computations. Nevertheless, the centralized approaches have complete knowledge, gathered from monitored data of distributed operator executions. Optimizing with respect to this knowledge may then result in a global optimum. For centralized approaches it becomes a necessity to apply heuristic techniques if the potential solution possibilities for placements grow.

3.4 Resource Optimization in other Disciplines

3.4.1 Elastic Processes

The execution of business processes in a Business Process Management System (BPMS) can include multiple resource-intensive tasks as stated by Schulte et al. [65]. Considering Section 2.3, this is very similar to DSP topologies, since data is processed and exchanged between services or operators respectively. Instead of optimizing operator resources, in

BPMS resources need to be provided to execute their instantiated processes on different services. In the following, approaches are summarized that aim to optimize the amount of provided resources, i.e., scaling resources.

Varying workloads for business processes ideally need to be handled in the Cloud to exploit resource, cost, and quality elasticity [65]. Therefore, the authors presented an elastic BPMS that is able to execute the processes in an elastic service environment hosted in the cloud. Scheduling service invocations among the Cloud requires to optimize the deployments. For this, the *Service Instance Placement Problem* is formulated by Hoenisch et al. [66, 67]. Herein, the cost for Virtual Machine (VM) provisioning and penalties, which accrue for late deployment and not complying to Service Level Agreements (SLAs), are minimized in an ILP optimization. The result is a cost-optimal placement of service instances.

In an extended approach, Hoenisch et al. [68] also take *Hybrid Clouds* into account. During continuous replacements of the service instances within the Cloud, migrations from a private Cloud to a public Cloud and vice versa are considered. These migrations lead to data transfer cost that have to be included in the ILP optimization model. The evaluation has shown that due to the reduction of more expensive inter-cloud data transfers and leased public Cloud VM instances, the cost for service placement decrease significantly.

In [69] the authors consider business processes that need to be executed according to user-defined non-functional requirements e.g., execution deadlines. A *Reasoner* component creates a scheduling plan that determines when a business process and its services are executed. Therefore, it has to be ensured that resources for execution are allocated in time. This is done by considering multiple time slots in the future which the service executions of the business processes are assigned to. Considering this scheduling plan, the resources in the Cloud are allocated such that cost and time are saved.

Euting et al. [70] present an optimization approach for scaling Cloud Resources based on fuzzy logic. The approach considers specified Key Performance Indicators (KPIs) as input values for scaling a BPM systems' provided VMs that are used for hosting involved services. These KPIs refer to e.g., the average complexity of started process instances and the average time for a process completion. The actual scaling decision is made in a *fuzzy controller* component that deduces the amount of VMs that need to be instantiated by applying pre-defined fuzzy logic rules. The fuzzy controllers' goal is to scale such that over- and underprovisioning of resources can be avoided and cost can be saved. Hence, the output of the fuzzy controller specifies the number of VMs to be instantiated at a given point in time.

3.5 State Migration

During operator movements, it has to be considered that its state needs to be migrated for each stateful operator. Current techniques in state migration refer to a pause-and-resume approach used in [16], whereas the literature also discusses a parallel track approach [71]. The former considers a shutdown or pause of the original operator before the state is migrated and efficiently restored on the new location. A drawback is the latency, which is caused by pausing an operator. The latter regards a parallel execution of both operators as long as they are synchronizing their states. Although it is faster than pause-and-resume, it requires enhanced mechanisms for state movement.

Ottenwälder et al. [43] focus on planning the migration of operators certain time steps ahead in order to minimize the network utilization. In some cases it is necessary to consider these migrations and their corresponding costs, especially when long-term benefits of a QoS-adequate network utilization can be achieved. So, e.g., when operators consider migrating if corresponding mobile data sources changed their location, a long-term reduction of network utilization is likely.

The up-front planning phase enables preparing the replacement by migrating parts of the operators a priori. To find an optimal solution, the authors use a probabilistic tree-like data structure that models possible migration possibilities with paths. The vertices indicate potential locations for the operators and its state, whereas the edges refer to migration actions between the vertices. The presented algorithm determines the migration that most likely results in a reduced network utilization by still ensuring end-to-end latency to be below a defined threshold.

To achieve migration of states, Distributed Data Stores (DDSs) can be used [16]. In DSP, this store acts as a repository for saving the states decoupled from the corresponding operators. Figure 3.6 shows an extended part of the architecture of Apache Storm, as presented by Cardellini et al. [16]. Worker processes, which run DSP operators, share their states together with other worker processes on a given supervisor node. In case of operator migration to another supervisor node, the DDS moves the corresponding states accordingly over the network.

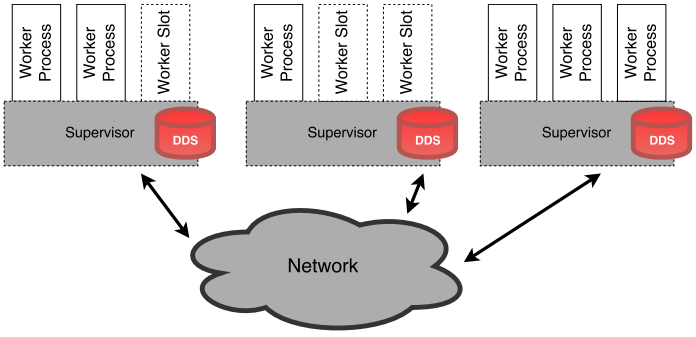


Figure 3.6: Distributed Data Stores [16]

CHAPTER 4

Requirements Analysis & Design

4.1 Required Functionality

In Section 1.4 we state that the VISP Ecosystem considers the *Reasoner* component for optimizing the placements of DSP topologies hosted by the VISP Runtime. For this, we propose the Optimal DSP Replacement (ODR) Reasoner which makes use of an ILP optimization model. The model includes parameters that are submitted by the VISP Runtime. After solving the optimization problem, placement instructions are returned to the corresponding VISP Runtime, which is in charge of deploying operators to the assigned hosts. With respect to optimality, we identify relevant QoS attributes that need to be optimized based on [10, 15].

- **Operator and network latency**
  This attribute influences the processing duration of DSP topologies. Hence, by optimizing placements such that operators are executed on resources with appropriate computing power, the performance can be maximized. Furthermore, the response time can be improved by avoiding long network distances that cause delays when data is streamed between the operators.

- **Enactment cost of using the underlying infrastructure**
  DSP operators can be deployed to decentralized hosts that provide their computing power at different cost or even with different cost models. Therefore, at given points in time the cost for enacting the DSP topology has to be re-evaluated and cost-optimal placements have to be computed.

- **Resource availability**
  Operators can be placed at different regions and hosts with varying availability. Especially in a heterogeneous environment like a Fog network, the differences in availability can be significant. Monitoring the online status of the involved

Cloud and Fog Resources is the basis for making optimal decisions to ensure the availability of the overall topology. Therefore, this data has to be considered in the optimization.

- **Cost of operator replacement**
  In order to optimize the criteria above, replacements have to be executed. These replacements lead to cost for e.g., temporary downtimes and data transfer. This has to be considered to limit the extra expenditure and save cost overall.

Beside these criteria, we take the heterogeneity of Fog and Cloud Computing resources into account. For this, we focus on different resource capabilities and varying network metrics. In contrast to the approach of Cardellini et. al [10], which considers an ILP Model that is partially reused in this work, we apply a dynamic optimization. In this context we describe dynamic optimization as iterative solving the placement problem over time. This enables continuous reconfigurations of operator (re)placements according to the current parameters measured in the system.

Figure 4.1 shows the optimization as cycle implemented by the ODR Reasoner based on the principles of the *PDCA cycle* and *Deming wheel* respectively [72]. These concepts include a control cycle with *Plan*, *Do*, *Check*, and *Act* phases. Mostly known from management sciences, herein this concept is used for providing an overview of the implemented optimization approach. By making use of the *PDCA cycle* we can illustrate the cyclic optimization behaviour which is responsible for reconfiguring operator placements.

The VISP Runtime is the operating environment for the DSP operators and initializes, i.e., plans, the optimization activities and considers the results for performing replacements of operators. The planning system is depicted in the top swim lane while the bottom swim lane (see Figure 4.1) refers to the operating system part. The control system is considered as the core component that is realized by the ODR Reasoner and depicted in the middle swim lane. For this, an optimization task (1) with a given set of optimization preferences (1c) is created first. The optimization task consists of the following attributes that are forwarded from the VISP Runtime to the ODR Reasoner to be used for executing the optimization:

- **DSP topology**
  This attribute contains the relevant information of DSP operators and their dependencies. For this, it describes the type of involved operators (e.g., source, sink, custom operator) and defines which deployment locations are allowed (see Section 4.4.3).

- **Resource infrastructure**
  The resource infrastructure is considered as the collection of all available deployment locations that can host DSP operators. In this work, we therefore take heterogeneous Fog and Cloud networks into account.

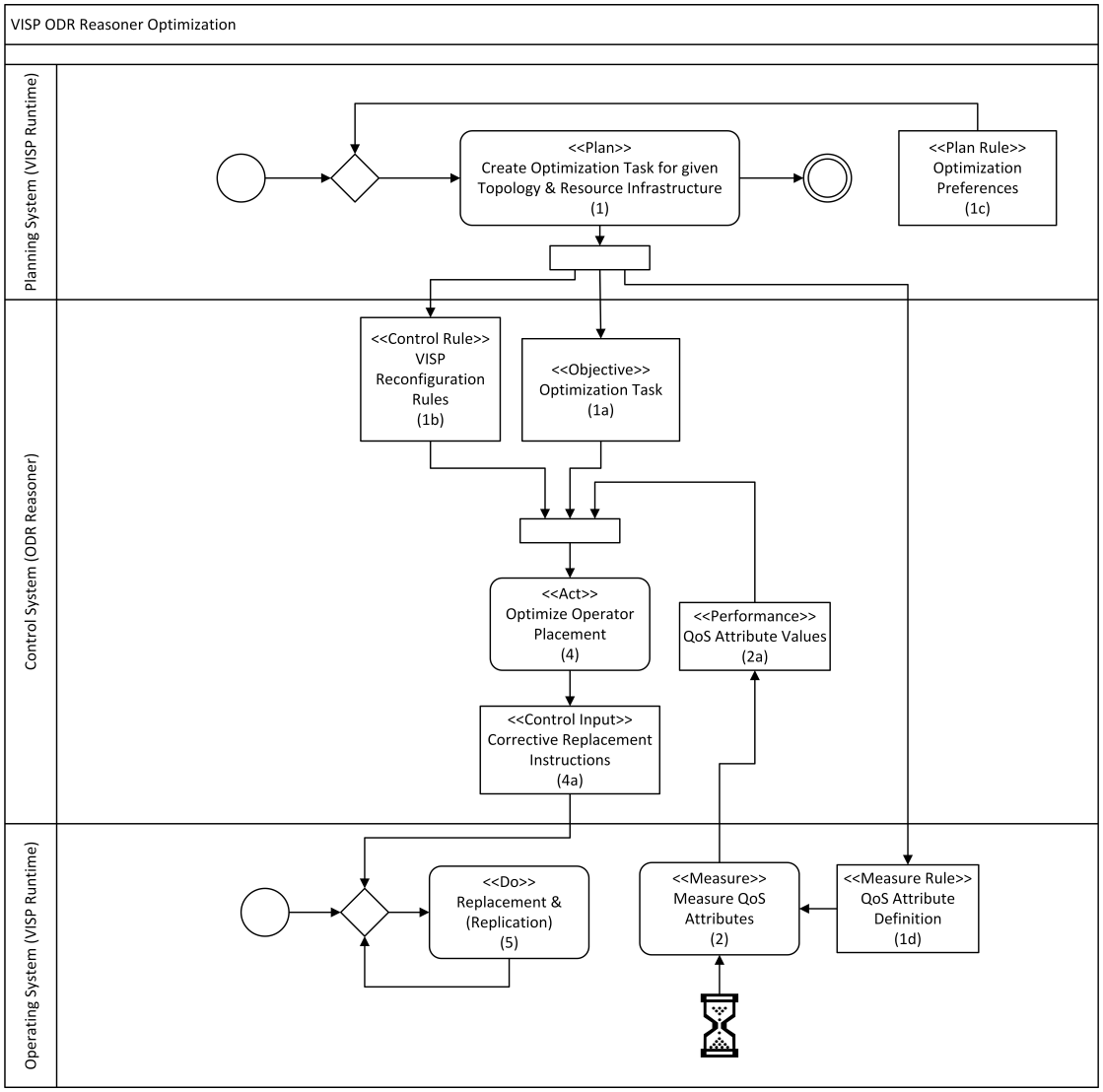


Figure 4.1: Plan-Do-Check-Act-based ODR

- **Monitoring data**
  The resource infrastructure comes with different capabilities and utilization. This information is collected to be used for optimization purposes.

- **Update Placeholder**
  The placeholder part is used for storing the updates of the attributes described above. It enables to track the changes of the resource infrastructure and corresponding monitoring data.

The optimization task is continuously maintained and updated by the ODR Reasoner as long as the optimization is executed. For that to happen, the ODR Reasoner stores the optimization task and incorporates monitored QoS attribute data. Additionally, the plan includes reconfiguration rules (1a) which tell the ODR Reasoner how the optimization has to be executed e.g., periodicity of placement reconfigurations or optimization goals like latency minimization. The reconfiguration rules are used to customize the optimization accordingly and to incorporate the requirements of the user. Third, QoS attributes are measured (2) in VISP and sent to the *Act* step (4) in order to incorporate them into the ILP optimization model. This data is used for computing parameters of the model on an ongoing basis. The *Optimize Operator Placement* action solves the placement problem and propagates the replacement instructions (4a) to the corresponding VISP Runtime (operating system) for execution (5).

4.2 System Model

To implement the optimization behaviour described in Section 4.1, we adapted the system and optimization model of Cardellini et al. [10], in the following referred to as *ODP*. At first, we extended the system model by aspects like resource limitation, processing duration of operators, and cost. The extension further incorporates a resource model that reflects the resource infrastructure information in several variables. Furthermore, the system model consists of variables that model the DSP topology. For this, we explain the notation of used variables in Table 4.1. Furthermore, we now highlight the main differences between *ODP* and our *ODR* extension.

4.2.1 Resource Model

The resource model is a graph G_{res}, containing computing nodes (i.e., hosts of operators) as vertices V_{res} and network links as edges E_{res}. The computing nodes are labelled with u and v respectively, considering $u, v \in V_{res}$. Corresponding QoS attributes characterize the current state of the network and its computing nodes. For this purpose, we consider the availability of computing nodes A_u and network links $A_{(u,v)}$ as main input to be able to optimize the overall topology availability. Similarly, the delay $d_{(u,v)}$ of a network link is incorporated as input for response time optimization. In contrast to Cardellini et. al [10], who focused on modeling response times and availabilities in their basic version, we also consider enactment cost C_u and migration cost $C_{(i,u,v)}$. To be specific on the resource node capabilities, we distinguish between different categories like the provided CPU frequency $P_{(CPU,u)}$, memory capacity $P_{(Mem,u)}$, storage capacity $P_{(HD,u)}$, and number of cores $P_{(Cores,u)}$. The speedup S_u of a concrete node $u \in V_{res}$ is a factor that indicates the potential of faster processing compared to a given reference processor. In addition, we model three resource classes with $SMALL$, $MEDIUM$, and $LARGE$ which contain resource nodes with similar capabilities and therefore corresponding speedups sp_{SMALL}, sp_{MEDIUM}, and sp_{LARGE}. So, if e.g., a computing node u is a *medium-sized* computing node and v is *small-sized* (i.e., $u \in MEDIUM$ and $v \in SMALL$) then it holds that u

Table 4.1: Notation Used for the Optimization

Symbol	Description
G_{dsp}	Graph representing the DSP topology
V_{dsp}	Operator set (vertices of G_{dsp})
E_{dsp}	Streams between operators (edges of G_{dsp})
G_{res}	Graph representing the resource infrastructure
V_{res}	Resource vertices (computing nodes) of G_{res}
E_{res}	Logical link set (edges of G_{res})
ET_i	Execution time (sec) per processed data tuple in operator $i \in V_{dsp}$
C_u	Enactment cost of computing node $u \in V_{res}$ per second
$C_{(i,u,v)}$	cost for migrating operator $i \in V_{dsp}$ from $u \in V_{res}^i$ to $v \in V_{res}^i$
$P_{(CPU,i)}$	Required CPU power (MHz) per core for operator $i \in V_{dsp}$
$P_{(Mem,i)}$	Required memory (MB) for operator $i \in V_{dsp}$
$P_{(HD,i)}$	Required storage (MB) for operator $i \in V_{dsp}$
$P_{(Cores,i)}$	Required number of cores for operator i $i \in V_{dsp}$
s_i	Size (MB) of image of operator $i \in V_{dsp}$
$T_{(actual,i)}$	Actual processing duration per tuple of operator $i \in V_{dsp}$
$T_{(max,i)}$	Maximum processing duration per tuple of operator $i \in V_{dsp}$
$P_{(CPU,u)}$	Available CPU power (MHz) in computing node $u \in V_{res}$
$P_{(Mem,u)}$	Available memory (MB) in computing node $u \in V_{res}$
$P_{(HD,u)}$	Available storage (MB) in computing node $u \in V_{res}$
$P_{(Cores,u)}$	Available number of cores in computing node $u \in V_{res}$
S_u	Processing speed-up of $u \in V_{res}$
S_i	Processing speed-up experienced by operator $i \in V_{dsp}$ in previous optimization period
$SMALL$	Set of computing nodes $u \in V_{res}$ with small resource capabilities
$MEDIUM$	Set of computing nodes $u \in V_{res}$ with medium resource capabilities
$LARGE$	Set of computing nodes $u \in V_{res}$ with large resource capabilities
sp_{small}	Speedup of computing nodes $u \in SMALL$
sp_{medium}	Speedup of computing nodes $u \in MEDIUM$
sp_{large}	Speedup of computing nodes $u \in LARGE$
A_u	Availability of computing node $u \in V_{res}$
$A_{(u,v)}$	Availability of $(u,v) \in E_{res}$
$d_{(u,v)}$	Network delay (sec) on $(u,v) \in E_{res}$
M	VISP Marketplace as separate node with $M \notin V_{res}$
$b_{(M,u)}$	Data rate (MB/s) between VISP Marketplace M and node $u \in V_{res}$
$x_{i,u}$	Decision variable for placement of $i \in V_{dsp}$ on V_{res}^i
$y_{(i,j)(u,v)}$	Decision variable for placement of $(i,j) \in E_{dsp}$ on $(u,v) \in E_{res}$
$x_{i,u}^{prev}$	Placement of $i \in V_{dsp}$ on V_{res}^i in the previous optimization period
$V_{res}^i \subseteq V_{res}$	Subset of nodes where $i \in V_{dsp}$ can be placed

runs faster than v (i.e., $S_u > S_v$). This categorization is made to limit the amount of provided resource classes, which are used for deployments, e.g., in the VISP Runtime. For retrieving new operator images from the VISP Marketplace, we model $b_{(M,u)}$ as data rate between the Marketplace M and node $u \in V_{res}$. Beside the current placement of an operator i on a specific node u that is modeled as decision variable $x_{i,u}$, we additionally consider the placement variable of the previous optimization period $x_{i,u}^{prev}$.

4.2.2 DSP Model

The DSP model reflects the structure of the DSP topology with all its involved operators and data streams. Therefore, the graph G_{dsp} models operators as vertices V_{dsp} and data streams, connecting the operators, as edges E_{dsp}. Similar to the resource supply, each operator $i \in V_{dsp}$ demands resources from different categories: CPU frequency $P_{(CPU,i)}$, memory capacity $P_{(Mem,i)}$, storage capacity $P_{(HD,i)}$ and number of cores $P_{(Cores,i)}$. Compared to ODP, the operator image size s_i is considered for migration purposes in ODR. The execution time that is needed to process k data tuples of operator $i \in V_{dsp}$ on a reference processor is modeled as ET_i. The number k results from the amount of tuples that is received from all operator predecessors when exactly one data tuple was injected by the source nodes (e.g., sensors). Therefore, according to the given topology we may consider more than one tuple (i.e., k tuples) for being processed in one processing step of an operator. In genereal, the execution time ET_i is used to describe the latency that is caused when operators execute their defined behavior. Additionally, $T_{(actual,i)}$ refers to the currently experienced tuple processing duration of operator i on the previous deployment location. To point out the main difference between ET_i and $T_{(actual,i)}$, we describe the former one as related to the present while the latter one is a monitored value from the past. Furthermore, $T_{(actual,i)}$ includes waiting times of tuples in queues before they can actually be processed. This is based on the architecture of VISP, which uses message queueing for tuple forwarding. Additionally, the user is able to limit $T_{(actual,i)}$ to a defined maximum duration of $T_{(max,i)}$. Finally, S_i stands for the speed-up that operator i has experienced at the deployment location in a previous optimization period. For this, it is assumed that the optimization is executed iteratively such that monitoring data of the past is available in current iterations.

4.3 Optimization Model

In literature, various techniques are used for solving the placement problem for DSP operators as discussed in Section 3.3.10. In this work we decided to make use of an ILP optimization model that is dynamically solved to remain at a global optimum. With dynamic solving, we refer to solve the optimization problem not only when the topology is deployed the first time, moreover we solve the optimization problem in fixed cycles repeatedly in the future. This enables to re-evaluate network structures and conditions that may have changed. Similar to the system model, we extend the ODP optimization model from Cardellini et al. [10]. Therefore, Section 4.3.1 presents the ODP optimization model with the objective function and its corresponding restrictions. In Section 4.3.2 we discuss the adaptations and extensions for the ODR model.

4.3.1 ODP Model

QoS Attributes

Response Time According to Cardellini et al. [10] there is no general definition for response time of a DSP system. Therefore, the authors express the response time as the maximum response time over all paths as described in Equation 4.1. For this, a path is a sequence of edges connecting multiple vertices. The response time of one path R_p is partitioned into two addends (Equation 4.2), whereas the first reflects the time spent for processing tuples within all operators along all paths and the second one considers the delay of data tuples within the network. For this, $R_i(\mathbf{x})$ in Equation 4.3 models the response time for a single operator $i \in V_{dsp}$ and $D_{(i,j)}(\mathbf{y})$ considers the delay between operator i and j with $i, j \in V_{dsp}$.

$$R = \max_{p \in \pi_{G_{dsp}}} R_p(\boldsymbol{x}, \boldsymbol{y}) \tag{4.1}$$

where

$$R_p(\boldsymbol{x}, \boldsymbol{y}) = \sum_{k=1}^{n_p} R_{i_k}(\boldsymbol{x}) + \sum_{k=1}^{n_p - 1} D_{(i_k, i_{k+1})}(\boldsymbol{y}) \tag{4.2}$$

$$R_i(\boldsymbol{x}) = \sum_{u \in V_{res}^i} \frac{ET_i}{S_u} x_{i,u} \tag{4.3}$$

$$D_{(i,j)}(\boldsymbol{y}) = \sum_{(u,v) \in V_{res}^i \times V_{res}^j} d_{(u,v)} y_{(i,j),(u,v)} \tag{4.4}$$

Availability The ODP approach models the DSP availability in Equation 4.5 by assuming independence of computing nodes $u \in V_{res}$ as well as network links $(i, j) \in E_{res}$. It is necessary to consider the logarithm of the DSP availability to avoid multiplications of decision variables $x_{i,u}$ and $y_{(i,j),(u,v)}$. This leads to linearity that is necessary for solving ILP problems, since quadratic or higher order equations cannot be solved with ILP solution procedures. Equation 4.8 describes the logarithm applied on Equation 4.5, with $\log(A_u(\mathbf{x})) = a_u$ and $\log(A_u(\mathbf{x}, \mathbf{y})) = a_{u,v}$. Although this application is not correct in all cases, it holds for the ODP model as shown in [10].

$$A(\boldsymbol{x},\boldsymbol{y}) = \prod_{i\in V_{dsp}} A_i(\boldsymbol{x}) \cdot \prod_{(i,j)\in E_{dsp}} A_{(i,j)}(\boldsymbol{y}) \tag{4.5}$$

where

$$A_i(\boldsymbol{x}) = \sum_{u\in V_{res}^i} A_u x_{i,u} \tag{4.6}$$

$$A_{(i,j)}(\boldsymbol{y}) = \sum_{(u,v)\in V_{res}^i \times V_{res}^j} A_{(u,v)} y_{(i,j),(u,v)} \tag{4.7}$$

$$\begin{aligned}\log A(\boldsymbol{x},\boldsymbol{y}) = &\sum_{i\in V_{dsp}} \sum_{u\in V_{res}^i} a_u x_{i,u} + \\ &+ \sum_{(i,j)\in E_{dsp}} \sum_{(u,v)\in V_{res}^i \times V_{res}^j} a_{(u,v)} y_{(i,j),(u,v)}\end{aligned} \tag{4.8}$$

Objective Function

The basic ODP model describes the objective function in Equation 4.9. Multiple objectives are included by making use of Simple Additive Weighting (SAW) [73]. This method considers objective functions with a sum of weighted terms, where each weight w_i with $i \in \{r, a\}$ is a parameter such that $\sum_i w_i = 1$ holds. Regardless of weighting parameters, it is necessary to balance the influence of each term to the objective function. Therefore, normalization of the QoS attributes is performed by dividing with $R_{\max} - R_{\min}$ and $\log A_{\max} - \log A_{\min}$ respectively. $R_{\max}$ and $R_{\min}$ are the maximum and minimum response times for the whole DSP topology. Analogously, the maximum and minimum availability is expressed with $\log A_{\max}$ and $\log A_{\min}$ respectively. The division results in equal scales ($[0, 1]$) for all QoS attributes.

$$F(\boldsymbol{x},\boldsymbol{y}) = w_r \frac{R_{\max} - R(\boldsymbol{x},\boldsymbol{y})}{R_{\max} - R_{\min}} + w_a \frac{\log A(\boldsymbol{x},\boldsymbol{y}) - \log A_{\min}}{\log A_{\max} - \log A_{\min}} \tag{4.9}$$

Optimization Problem Formulation

The basic linear program can be formulated as shown in Equations 4.10 - 4.18. Compared to the objective function in Equation 4.9, $F(\boldsymbol{x},\boldsymbol{y})$ is replaced with $F'(\boldsymbol{x},\boldsymbol{y},r)$ to avoid nonlinearity as introduced with $R = \max_{p\in\pi_{G_{dsp}}} R_p$ in Equation 4.1. The used auxiliary variable r replaces $R(\boldsymbol{x},\boldsymbol{y})$ in the objective function. Additionally, the constraint presented in Equation 4.11 ensures that r is greater or equal than the response times of all topology paths. Considering also that in the optimum r has to be minimized, we have $r = \max_{p\in\pi_{G_{dsp}}} R_p(\boldsymbol{x},\boldsymbol{y}) = R(\boldsymbol{x},\boldsymbol{y})$, which is therefore a valid replacement.

$$\max_{\boldsymbol{x},\boldsymbol{y},r} F'(\boldsymbol{x},\boldsymbol{y},r)$$

where

$$F'(\boldsymbol{x},\boldsymbol{y},r) = w_r \frac{R_{\max} - r}{R_{\max} - R_{\min}} + w_a \frac{\log A(\boldsymbol{x},\boldsymbol{y}) - \log A_{\min}}{\log A_{\max} - \log A_{\min}} \tag{4.10}$$

subject to:

$$r \geq \sum_{k=1}^{n_p} \sum_{u \in V_{res}^{i_k}} \frac{ET_i}{S_u} x_{i_k,u} + \sum_{k=1}^{n_p-1} \sum_{(u,v) \in V_{res}^{i_k} \times V_{res}^{i_{k+1}}} d_{(u,v)} y_{(i_k,i_{k+1}),(u,v)} \quad \forall p \in \pi_G \tag{4.11}$$

$$\sum_{i \in V_{dsp}} P_i x_{i,u} \leq P_u \quad \forall u \in V_{res} \tag{4.12}$$

$$\sum_{u \in V_{res}^i} x_{i,u} = 1 \quad \forall i \in V_{dsp} \tag{4.13}$$

$$\sum_{i \in V_{dsp}} x_{i,u} \leq 1 \quad \forall u \in V_{res} \tag{4.14}$$

$$x_{i,u} = \sum_{v \in V_{res}^j} y_{(i,j),(u,v)} \; \forall (i,j) \in E_{dsp},\; u \in V_{res}^i \tag{4.15}$$

$$x_{j,v} = \sum_{u \in V_{res}^i} y_{(i,j),(u,v)} \; \forall (i,j) \in E_{dsp},\; v \in V_{res}^j \tag{4.16}$$

$$x_{i,u} \in \{0,1\} \quad \forall i \in V_{dsp}, u \in V_{res}^i \tag{4.17}$$

$$y_{(i,j),(u,v)} \in \{0,1\} \; \forall (i,j) \in E_{dsp}, (u,v) \in V_{res}^i \times V_{res}^j \tag{4.18}$$

The resource limitation constraint, shown in Equation 4.12, is used in the ODP model to ensure that the demanded resources of operator $i \in V_{dsp}$ are less than the provided resources of node $u \in V_{res}$ which the operator is deployed to. In the ODR model we drop this constraint. Instead, we distinguish between specific resource categories as CPU, memory and storage as presented later in Section 4.3.2. Equations 4.13 and 4.14 restrict one operator to be exactly placed at one resource node and one resource node is restricted to host up to one operator respectively. The logical *AND* constraint $y_{(i,j),(u,v)} = x_{i,u} \wedge x_{j,v}$ is modeled in Equations 4.15 and 4.16. As described in Table 4.1, decision variable $y_{(i,j),(u,v)}$ indicates if the logical network link between nodes $u \in V_{res}^i$ and $v \in V_{res}^j$ is used by operators $i, j \in V_{dsp}$. Therefore, the logical *AND* constraint equations ensure that if $y_{(i,j),(u,v)}$ evaluates to one it has operator i and j to be deployed on nodes u and v respectively. Finally, Equations 4.17 and 4.18 ensure that decision variables $x_{i,u}$ and $y_{(i,j),(u,v)}$ are boolean variables which reflect the placement decision after optimization.

4.3.2 Extensions in the ODR Optimization Model

We now present the changes and extensions made with respect to the ODP optimization model of Cardellini et al. [10]. The ODR model also considers cost- and migration-related aspects and makes further assumptions, e.g., new QoS attributes are introduced and existing equations are updated.

QoS Attributes

Enactment Cost To account for cost that are caused by enacting DSP topologies on the resource infrastructure, we model $C_{op}(\boldsymbol{x})$ as total enactment cost per second as shown in Equation 4.19.

$$C_{op}(\boldsymbol{x}) = \sum_{i \in V_{dsp}} \sum_{u \in V_{res}^i} C_u x_{i,u} \tag{4.19}$$

Migration Cost We consider periodic reconfiguration cycles to apply operator replacements. For this, we take into account that cost for migrating operators from an old deployment location to a new one accrue. Technically this is performed by shutting down existing operators and pull the corresponding operator images from the VISP Marketplace to instantiate new operators on determined optimal locations. The overall migration cost, as shown in Equation 4.20, is the sum of all single migration cost $C_{(i,u,v)}$ that result from a planned migration (i.e., $x_{i,u}^{prev} = x_{i,v} = 1$). This cost is described in Equation 4.21 by considering the operator image size s_i and the data rate $b_{(M,v)}$ for pulling new operator images. The division of these two variables yields the duration that is needed to load the image to destination node $v \in V_{res}^i$ while operator i is shutting down. Multiplying this duration with C_u reflects the value which would be invested in executing operator i on node u for the ongoing migration. We see this value as utility value which is missing in case of operator replacement and hence it is a good proxy for migration cost valuation.

$$C_{mig}(\boldsymbol{x}) = \sum_{i \in V_{dsp}} \sum_{u \in V_{res}^i} \sum_{v \in V_{res}^i \setminus \{u\}} C_{(i,u,v)} x_{i,u}^{prev} x_{i,v} \tag{4.20}$$

$$C_{(i,u,v)} = \frac{s_i}{b_{(M,v)}} C_u \tag{4.21}$$

Response Time In general we stick to the response time of the ODP model from Equation 4.1 - 4.4, but we further refine the speedup used in Equation 4.3. To account for specific resource nodes, where the operators are deployed to, the speedup S_u is assigned with one of three predefined speedups according to the resource category of node $u \in V_{res}$.

$$S_u = \begin{cases} sp_{small} & u \in SMALL \\ sp_{medium} & u \in MEDIUM \\ sp_{large} & u \in LARGE \end{cases} \tag{4.22}$$

ODR: Objective Function

Making use of the SAW technique, we add two additional weighted cost terms to the objective function from Equation 4.10 to obtain the ODR objective function as shown in Equation 4.23.

$$\begin{aligned} F'_{cost} &= F'(\boldsymbol{x}, \boldsymbol{y}, r) + \\ &+ w_{c_{op}} \frac{C_{op\,\max} - C_{op}(\boldsymbol{x})}{C_{op\,\max} - C_{op\,\min}} + w_{c_{mig}} \frac{C_{mig\,\max} - C_{mig}(\boldsymbol{x})}{C_{mig\,\max} - C_{mig\,\min}} \end{aligned} \tag{4.23}$$

Constraints

Budget Constraints To limit the amount of enactment cost, parameter B_{op} is considered in the introduced constraint shown in Equation 4.24. According to Woodside et al. [74], deployment changes (e.g., operator replacements) should be limited to save cost and keep the performance at a certain level. For this, the authors state that penalizing these changes is a promising technique. In our approach we achieve this by taking migration cost into account. Furthermore, by introducing constraints as e.g., shown in Equation 4.25, a rule-based approach [74] can limit the number of deployment changes.

$$C_{op}(\boldsymbol{x}) \leq B_{op} \tag{4.24}$$

$$C_{mig}(\boldsymbol{x}) \leq B_{mig} \tag{4.25}$$

Processing Duration Constraints Since the optimization should ensure performance by executing replacements, we introduce another change trigger for new placement decisions. This is formulated as constraint considering the processing duration of tuples as shown in Equation 4.26. It enables that operators are deployed to hosts with more resource capacities (i.e., higher speed-up S_u) if the processing duration of the current optimization run exceeded the expected maximum duration. Especially, when preceding operators send many tuples to an already overloaded operator, the processing duration increases even more. In this case, a more powerful resource node of a higher category (e.g., *medium* or *large*) can be considered as new deployment location to decrease the queued tuples by processing it faster. The left-hand side of Equation 4.26 is the process

duration that results if operator i is deployed to resource node u with speedup S_u. This term has to be smaller than the user defined limit $T_{max,i}$.

$$\sum_{u \in V_{res}} T_{(actual,i)} \frac{S_i}{S_u} x_{i,u} \leq T_{(max,i)} \quad \forall \in V_{dsp} \tag{4.26}$$

Constraint Extensions to Equation 4.12 As a substitute for the general resource capability constraint in Equation 4.12, we consider the set of Equations 4.27 - 4.29. First, 4.27 ensures that the required CPU power of the deployed operator is smaller than the provided one. In this case, we consider CPU power as the product of the number of cores and CPU frequency. This enables us e.g., to deploy operators with highly required CPU frequency also to nodes that have lower CPU frequency but are equipped with more cores. Likewise, we assume that higher CPU frequency can offset a shortage of cores. Equation 4.28 deals with memory capacity that has to be guaranteed on resource nodes to be able to host operators. The storage capacity constraint in Equation 4.29 is modeled analogously.

$$\sum_{i \in V_{dsp}} P_{(CPU,i)} P_{(Cores,i)} x_{i,u} \leq P_{(CPU,u)} P_{(Cores,u)} \quad \forall u \in V_{res} \tag{4.27}$$

$$\sum_{i \in V_{dsp}} P_{(Mem,i)} x_{i,u} \leq P_{(Mem,u)} \quad \forall u \in V_{res} \tag{4.28}$$

$$\sum_{i \in V_{dsp}} P_{(HD,i)} x_{i,u} \leq P_{(HD,u)} \quad \forall u \in V_{res} \tag{4.29}$$

4.4 Software Design: ODR Reasoner

To implement the ODR optimization, we designed the corresponding VISP ODR Reasoner that interacts with the VISP Runtime. According to the DSP placement approach criteria of Lakshmanan et al. [64], ODR can be characterized as a loosely-coupled component that runs as single service to serve the DSP system with placement instructions. It collects monitoring metrics of the available Cloud and Fog Resources and the latency among them. Aiming for a global optimum with respect to the identified QoS criteria, periodic re-evaluation of the situation in the network and a corresponding reconfiguration process of DSP placements are required. These placement updates consider to move existing operators to new locations. Furthermore, scaling of provided operator instances is performed to avoid potential over- and underprovisioning of DSP resources.

The following sections present the static design of the main system building blocks as well as the dynamic behaviour of the implemented optimization approach.

4.4.1 Static View

The main system components are depicted in Figure 4.2. The *ODR Reasoner* is the overall service that consists of several sub-components. These components exchange optimization requests and their parameter with the *VISP* Runtimes (i.e., one-to-many VISP Runtime instances). Therefore, the northbound interface realized by the *VISP ODR Reasoner API* (in the following referred to as API) is used for receiving optimization requests from the Runtime. The southbound interface between the *Reconfiguration Manager* and VISP is used for returning placement instructions. Considering the second client, the *Simulator* acts as a surrogate for VISP in case when the ODR Reasoner has to be tested.

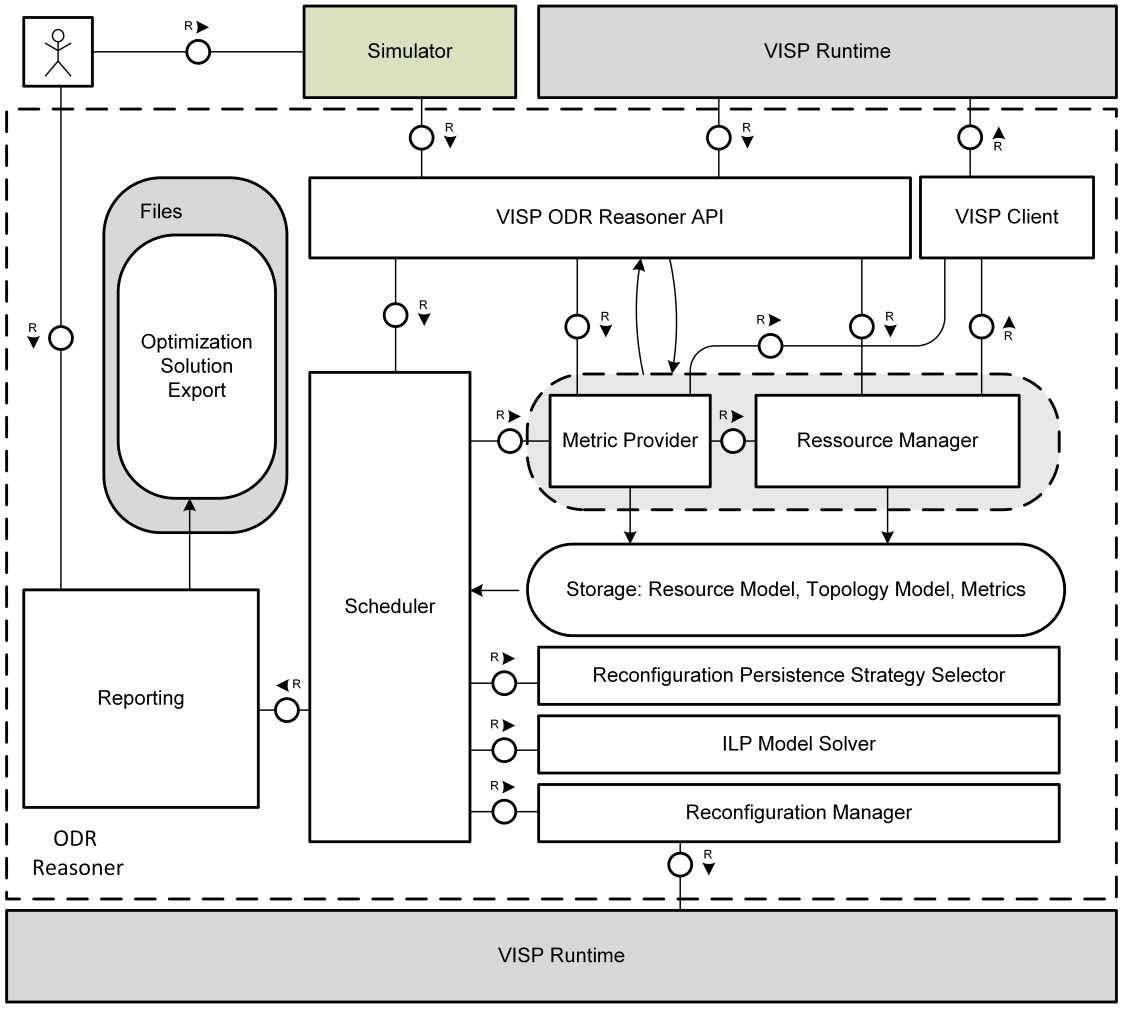


Figure 4.2: System Components (Block Diagram)

Focusing on the ODR Reasoner, the *API* is considered as a service endpoint that creates a customized *Metric Provider* as well as a *Resource Manager* for each optimization

task that is received from the clients. In general, an optimization task is considered as the major data structure including the attributes described in Section 4.1. The *Metric Provider* is responsible for deriving the optimization model parameter, as mentioned in the DSP model (see Section 4.2.2). The parameters are used to build V_{dsp}. Therefore, it considers metrics that are either initially received or requested from the VISP Runtime with the help of *VISP Client*. Analogously, the resource model (see Section 4.2.1) is built with the Resource Manager. It creates an internal representation of the resource infrastructure as graph V_{res}. To save the graphs, metrics, optimization settings and placement recommendations, we consider to encapsulate this data into the corresponding optimization task and persist it to an in-memory *Storage*.

The *Reconfiguration Persistence Strategy Selector* is used for achieving partial persistence for the optimized placement solutions over time. According to Woodside et al. [74], deployment persistence refers to a limitation in deployment changes (e.g., operator replacements) since frequent adaptations are costly (e.g., possible downtime cost) and may lead to an inefficient utilization of resources. Hence, the *Reconfiguration Persistence Strategy Selector* is in charge to constrain the amount of solution possibilities to a certain extent. The actual solution of the placement problem is then computed by an *ILP Model Solver*. It makes use of a ILP solver that returns the placement results. The *Reconfiguration Manager* then transforms the results to a file that can be uploaded to any *VISP* Runtime for applying the computed changes. *Reporting* is subsequently invoked to visualize the changes and to observe the development of QoS attributes. Furthermore, it logs the QoS metrics to the file system. This can be used later for in-depth analysis and evaluation of the results. The metrics are derived from the measured *VISP* Runtime monitoring data. For this, the *Metric Provider* and *Resource Manager* are used to request the data periodically. To support this, the *VISP Client* component is invoked to access the updates of the *VISP* Runtimes.

Finally, we identify the *Scheduler* as central component that controls the optimization in a cyclic manner. It invokes the previously mentioned components to execute the saved optimization tasks. Technically, multiple optimization tasks can be handled by the Scheduler in parallel. Hence, numerous DSP topologies can be optimized by ODR at the same time.

4.4.2 Dynamic View of the System Interaction

We now focus on the dynamic behaviour between the VISP Runtimes and the ODR Reasoner. For this, we first abstract from internal ODR components to investigate the cross-system interaction. Overall, three interaction use cases can be distinguished. First, we consider one VISP Runtime to add a new optimization task to the ODR Reasoner. Second, in order to receive continuing operator replacement instructions for a given DSP topology, we consider the starting action of a previously added optimization task. For the sake of completeness, there also exists a deletion possibility for optimization tasks. We now describe the interactions of the first and second use case by using UML sequence diagrams as shown in Figures 4.3 and 4.4.

Add Optimization Task

Figure 4.3 depicts the interaction between one VISP Runtime and the ODR Reasoner. An optimization task can be added by sending a *VISPRuntimeCallback* that identifies the issuing VISP Runtime instance. This information is then used to pull the required environment information (i.e., topology, resource infrastructure) in subsequent requests. A task identifier *TaskID* is generated and assigned to the optimization task. The *TaskID* is also returned to the VISP Runtime for being able to perform further transactions on the task. Additionally, preferences like the optimization cycle period are loaded from the VISP Runtime to control the optimization behaviour. If finally all data is requested successfully, parsing is started to map the data to internal data structures. Furthermore, the parametrization step refers to the process of deriving parameters from requested data as defined in the system model and described in detail in Section 4.4.4.

Figure 4.3: Sequence Diagram - Add an Optimization Task

Resource Pools The request *getResourcePoolInfo()* uses the concept of *Resource Pools*. *Resource Pools* are considered in the VISP Runtime to host parts of a DSP topology. These pools are realized based on e.g., VMs. Each *Resource Pool* considers at least one but typically multiple VMs that can host at least one operator in a container. Therefore, the ODR Reasoner does not directly receive a resource infrastructure as modeled with G_{res} in the system model, moreover it has to consider *Resource Pools*. This requires further transformations of the *Resource Pool* information to internal data structures (i.e., G_{res}) which is described in Section 4.4.3.

Start Optimization Tasks

To actually start the added optimization task, the issuing VISP Runtime has to send the *startOptimization(TaskID)* request as depicted in Figure 4.4. The ODR Reasoner then loads the optimization task with the given *TaskID* from the internal storage and initiates its execution.

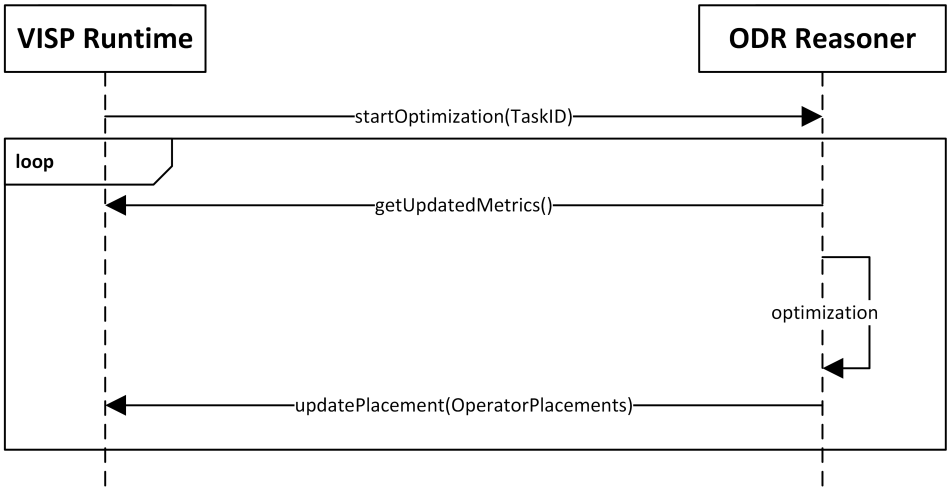


Figure 4.4: Sequence Diagram - Start an Optimization Task

Next, the execution is considered to run in a loop according to the specified optimization cycle period. Metrics with respect to the operators and the resource infrastructure get updated before starting the actual optimization, as will be described in detail in Section 4.4.4. Finally, after the results can be provided, the placement instructions are sent to one VISP Runtime via *updatePlacement(OperatorPlacements)*. Technically, VISP Runtimes consider a specific file that contains the topology placement information, as we will describe in the following section.

4.4.3 VISP Integration

The main use cases that consider the interaction between VISP Runtimes and the ODR Reasoner use different data structures. The data from all currently running VISP Runtimes is required to build the internal system model as described in Sections 4.2.1 and 4.2.2. Therefore, we subsequently discuss the data formats and transformations that are necessary. In general, the data is transferred by making use of Representational State Transfer (REST) interfaces of the involved systems.

VISP Topology Description Language

Hochreiner et al. [11] designed the *VISP Topology Description Language* in order to ease the communication between VISP and other entities that download/upload DSP topologies from/to a VISP Runtime instance. Listing 4.1 includes a sample topology

description consisting of a data source, two operators, and a sink. These components define a simple sequential topology, that processes temperature data within a manufacturing context.

Listing 4.1: Simple Temperature Data Processing Topology

```
1  $source = Source() {
2   concreteLocation = 128.120.172.182/cloudPool0,
3   type             = "TemperatureSensor",
4   outputFormat     = "temperatureData"
5  }
6
7  $step1 = Operator($source) {
8   allowedLocations = 128.120.172.182/*,
9   concreteLocation = 128.120.172.182/cloudPool0,
10  inputFormat      = "temperatureData",
11  type             = "CleanData",
12  outputFormat     = "cleanedTemperatureData",
13  size             = "medium"
14 }
15
16 $step2 = Operator($step1) {
17  allowedLocations = 128.120.0.1/fogPool1 128.120.0.1/fogPool2,
18  concreteLocation = 128.120.0.1/fogPool1,
19  inputFormat      = "cleanedTemperatureData",
20  type             = "EnrichWithMachineInfo",
21  outputFormat     = "TemperatureOfManufacturingMachine",
22  size             = "large"
23 }
24
25 $log = Sink($step2) {
26  concreteLocation = 128.120.172.182/cloudPool0,
27  inputFormat      = "TemperatureOfManufacturingMachine",
28  type             = "logOperator",
29 }
```

The source considers temperature data that is emitted by a sensor. The sensor is deployed to the location defined by the IPv4 address of the *concreteLocation* attribute. It is identified by the attribute *type* and considers the specified *outputFormat*, which is defined as *temperatureData*. The *size* attribute with its possible values *small*, *medium*, and *large* enables to scale operator instances. For this, *small* requires to start exactly one instance, whereas two are instantiated for *medium* and four for *large*. As defined for the *step1* operator, the attribute *allowedLocations* defines a set of locations that can be used for hosting the specified operator. During optimization of the ODR Reasoner, the placement approach uses these restrictions to avoid invalid placements on the resource infrastructure.

Resource Pool Mapping

As discussed earlier, VISP considers Resource Pools that are managed by VISP Runtimes. In order to transform these resource infrastructure format to an internal graph representation G_{res}, we map each used Resource Pool to a subset of the corresponding vertices V_{res}.

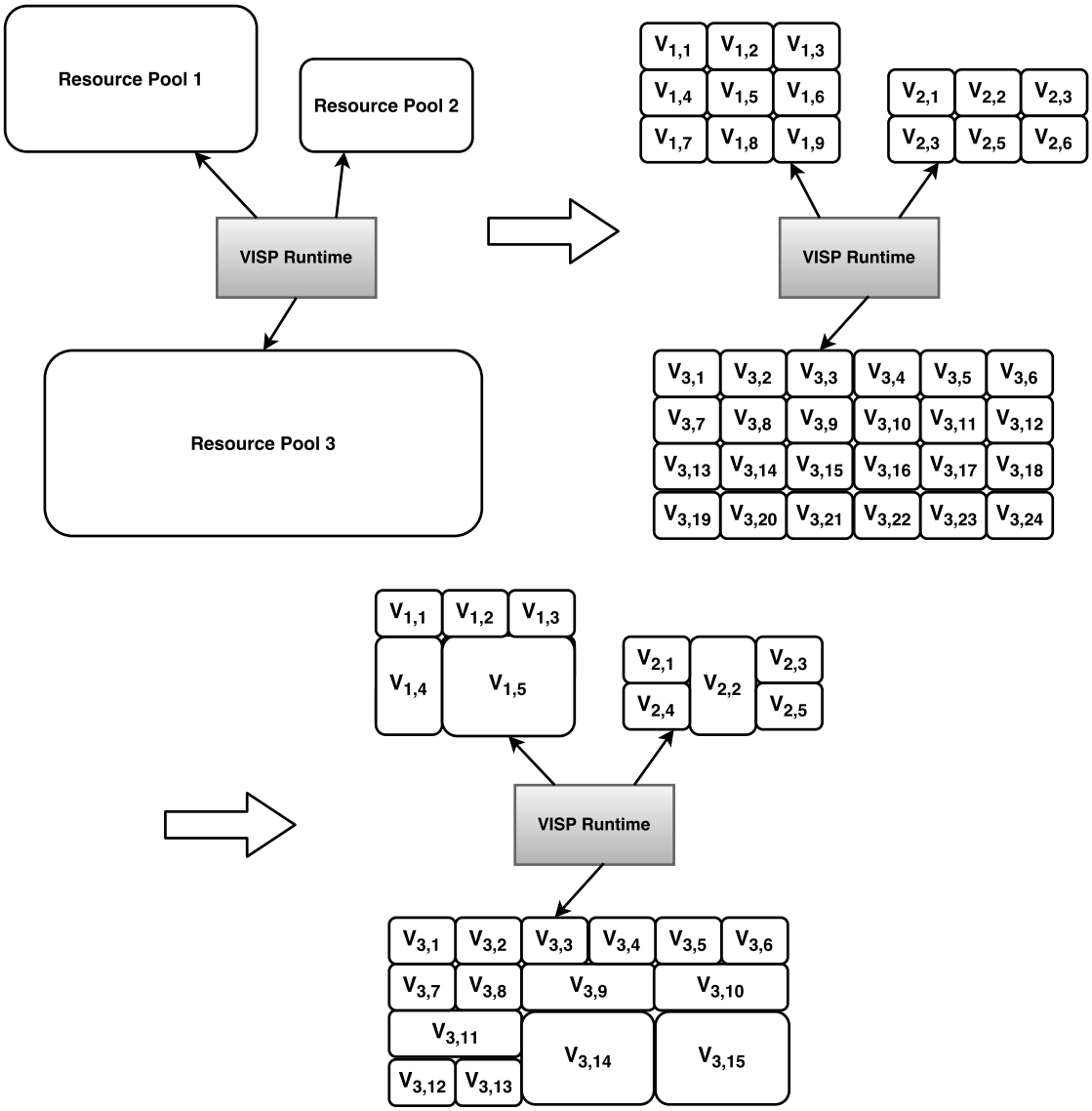


Figure 4.5: Resource Pool Mapping

Figure 4.5 depicts the mapping process of Resource Pools. Initially, we consider three Resource Pools of different sizes that are controlled by a VISP Runtime. In this visualization, the size of a Resource Pool refers to the resource capacities like CPU frequency & cores,

memory, and storage. After the first mapping step, each pool is partitioned into equal-sized resource nodes. In ODR we consider a node to be able to host one container which runs an operator. Every node has the capacity of a specified reference host (e.g., 2400 Mhz, 0.2 cores, 512 MB memory, and 300 MB storage). Due to the fact that we consider to place operators with different resource demands on these nodes, the provided resource capacity of one node might not suffice (e.g., when operators have to bear higher loads). Therefore, nodes with more computational power and storage are required. This can be achieved by mapping the equally partitioned Resource Pools to partitions depicted in the third step of Figure 4.5. For this, resource nodes of three different sizes can be used to place operators. We consider *small*, *medium*, *large* nodes, whereas each category of nodes uses a fixed portion of the overall capacity. By default, we set this portion to a third. We consider a *medium* sized node to have twice the capacity than the reference host which is considered as *small* node. Analogously, *large* nodes have the fourfold capacity of *small* nodes. If *large* or even *medium* nodes would demand more than the assigned portion, we add *small* nodes accordingly, such that the overall capacity of the Resource Pool is used.

First, the mapping process results in the creation of resource vertices V_{res}, i.e., nodes, as discussed above. In order to build the corresponding edges E_{res} to complete the construction of G_{res}, we consider a complete graph. For this, it holds that $E_{res} = \{\{u_j, u_k\} : 1 \leq j < k \leq n\}$ with $V_{res} = \{u_1, ..., u_n\}$. So, each constructed node is connected to all other nodes.

As it can be seen, there is a connection to the *size* attribute of VISP Topology Description Language files. For this, it has to be mentioned that after the placements are optimized, the results are uploaded to a VISP Runtime. So, if the placement results in deploying a given operator on e.g., a *medium* node, the file is created with the *size* attribute set to *medium*. Therefore, the provided resources are doubled by hosting two operators, which we assume to behave similar to providing only one operator but with doubled power. This is performed analogously with *large* operators. Furthermore, the uploaded topology file contains the placement information that is stored in the *concreteLocation* attribute. This information is propagated to the VISP Runtime for executing the updates within the topology placement.

Multiple VISP Runtimes We further consider the case that more than one VISP Runtime can manage Resource Pools for a given DSP topology. For this, operators of the topology are deployed to Resource Pools of different involved Runtimes. Thus, the distributed organization requires to perform the mapping for all Resource Pools as described above for single VISP Runtime case. For this, the relevant information of Resource Pools is collected by requesting the involved VISP Runtimes. The ODR Reasoner then applies the Resource Pool mapping process for each Runtime sequentially. Finally, the resulting graphs are connected to have exactly one complete graph to be considered as G_{res}.

4.4.4 Dynamic View of the ODR Core Components

To not only reflect the high-level interactions between VISP Runtimes and the ODR Reasoner, we now analyze the design of the internal behaviour of involved ODR system components.

Add Optimization Tasks

We now consider the dynamic view of adding an optimization task to the ODR Reasoner. For this, we consider three involved ODR components (API, Metric Provider, and Data Context) as depicted in Figure 4.6.

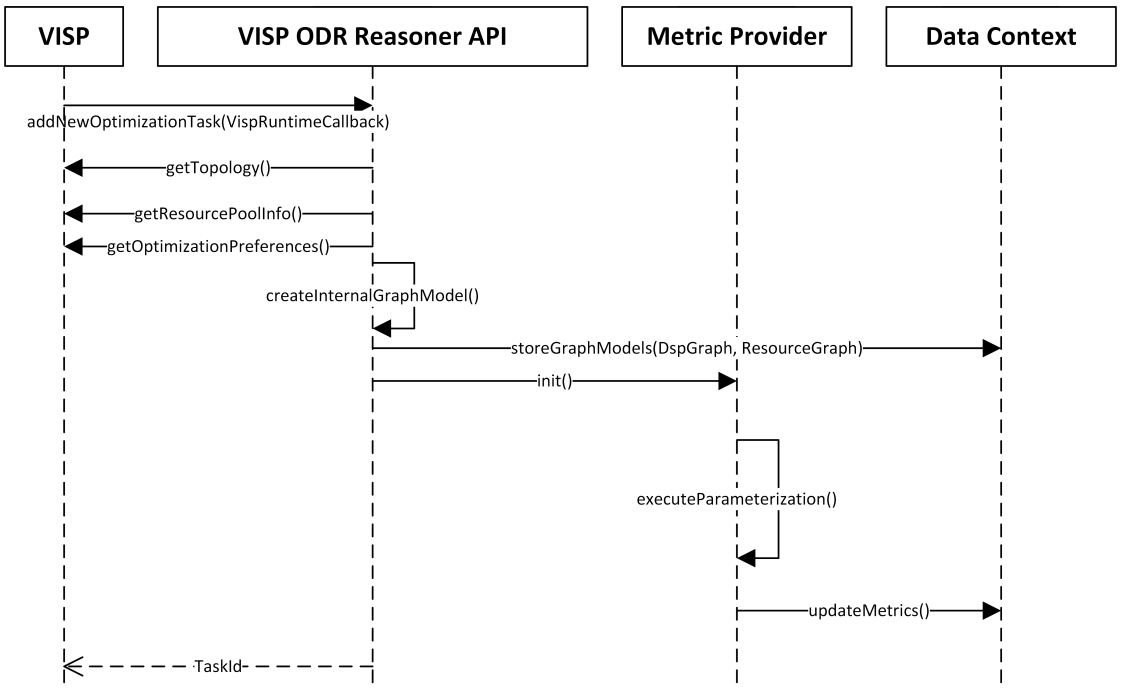


Figure 4.6: Add one Optimization Task

The first four interactions between a VISP Runtime and the API have already been shown in Section 4.4.2. Considering the requested DSP topology, Resource Pools and optimization preferences, the ODR reasoner can create internal representations according to the graphs G_{res} and G_{dsp} of the system model. The Data Context is used for storing the created graph models, that are subsequently initialized by the Metric Provider (see Section 4.4.1). This further initialization is necessary, because not all information can be directly requested from the VISP Runtimes at the beginning. Instead it needs parameter estimation or additional requests. Therefore, the Metric Provider triggers a parametrization process that assigns values to the following variables used in the optimization model:

- Response time boundaries: $R_{\max}$, $R_{\min}$
  estimated by maximizing/minimizing $R(\boldsymbol{x}, \boldsymbol{y})$
- Availability boundaries: $A_{\max}$, $A_{\min}$
  estimated by maximizing/minimizing $A(\boldsymbol{x}, \boldsymbol{y})$
- Enactment cost boundaries: $C_{op\,\max}$, $C_{op\,\min}$
  estimated by maximizing/minimizing $C_{op}(\boldsymbol{x})$
- Migration cost boundaries: $C_{mig\,\max}$, $C_{mig\,\min}$
  estimated by maximizing/minimizing $C_{mig}(\boldsymbol{x})$
- Network delay $d_{(u,v)}$ for all $(u, v) \in E_{res}$
  requested from all VISP Runtimes that maintain Resource Pools
- Speedup S_u for all $u \in V_{res}$
  assigned value that corresponds to the size of the created node u (*small*, *medium*, *large*)
- Execution time ET_i for all $i \in V_{dsp}$
  either assigned a user-defined default value or estimated by applying a specific testing process as described in Section 4.4.5

The remaining parameters that are defined in the system model can be directly derived from the requested data in the *createInternalGraphModel()* step. The *updateMetrics()* step is then executed to store the parametrized model in the Data Context. Finally, the *TaskID* is returned.

Starting and Running an Optimization Task

After adding an optimization task, its periodic execution is triggered from the issuing VISP Runtime as modeled in Figure 4.7. The Scheduler first loads the parametrized graph models from the Data Context. In order to restrict the number of operator replacements, the Persistence Strategy component is invoked. For this, operators are either pinned to a given Resource Pool or refer to the use migration cost which should be minimized. Considering this minimization, the extent of changing placements is limited inherently in the optimization problem as it can be seen in Equation 4.23. The graph models as well as the optimization parameters (i.e., optimization weights from Equations 4.10 and 4.23) are passed to the ILP Model Solver. This component optimizes the operator placements that can be propagated to the Reconfiguration Manager. To ensure that the results are returned to the correct VISP Runtime, the corresponding VISP Runtime Callback is loaded. Finally, a VISP Topology Description Language file is created in the Reconfiguration Manager and uploaded to the VISP Runtime by using the network address and port defined in the VISP Runtime Callback.

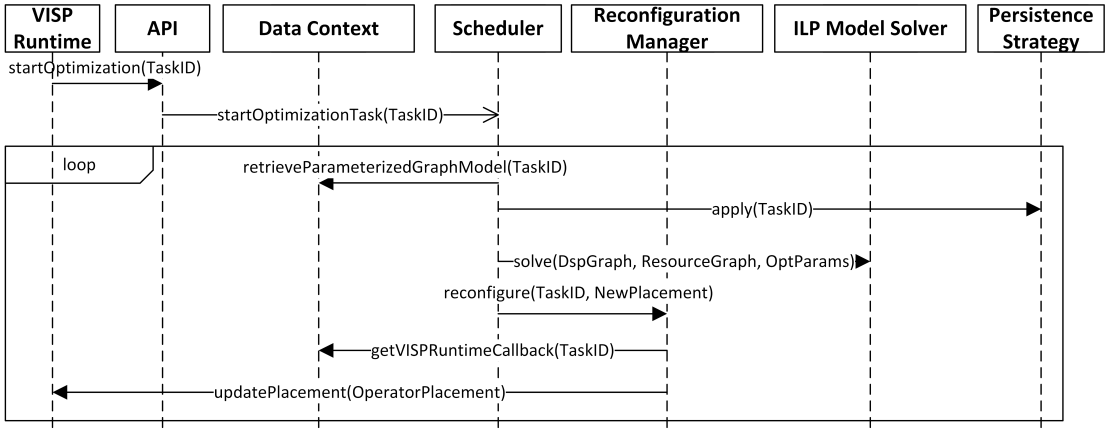


Figure 4.7: Starting and Running an Optimization Task

Update Resources and Metrics

In the previous use case we consider the graph model already to be updated and stored in the Data Context. Now, we describe how the resource model and DSP model in form of G_{res} and G_{dsp} respectively, are updated throughout the life time of an optimization task. In Figure 4.8, the Scheduler initiates the update cycle for a given optimization task identified with the *TaskID* by requesting the assigned Metric Provider. Herein, we consider that every optimization task has its own Metric Provider and Resource Manager. This is because different optimization tasks can have divergent settings that apply in the update phase, e.g., some optimization tasks require to update in short time cycles whereas others consider longer ones. So, the Metric Provider and Resource Manager together with the help of the VISP Client build an interface between the origin of the optimization task that parses the relevant information to an optimization task that can be processed by the ODR Reasoner. The first call, made by the Metric Provider, is to retrieve the current metrics update from VISP Runtimes. This comprises the DSP topology and its monitoring data for the hosted operators. The Metric Provider further invokes the assigned Resource Manager to update the resource infrastructure information. For this, it requests the resource infrastructure graph and executes the Resource pool mapping process as described in Section 4.4.3. Additionally, the Resource Manager checks if new VISP Runtimes and/or Resource Pools have been added or removed. If so, the Resource Pool mapping process updates the graph models G_{res} and G_{dsp} correspondingly. In the *recomputeParameter(...)* step, the parametrization process of Section 4.4.4 is started to update the remaining variables. The fully updated model is then stored in the Data Context again to be used for further placement optimizations.

4.4.5 Parametrization Phases of the Optimization Model

In order to know when parameters of the system model (see Section 4.2) are assigned to values, we provide an overview in Table B.4. Therefore, we distinguish between three

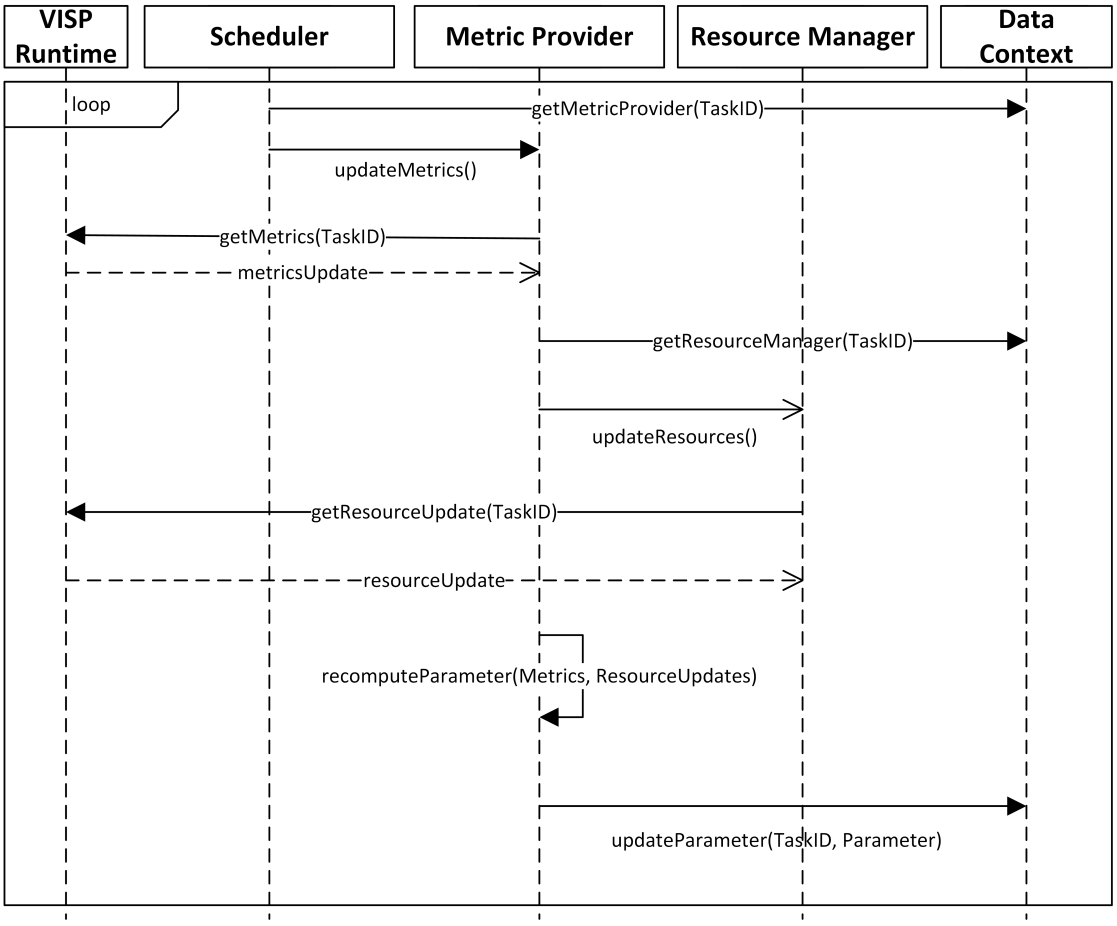


Figure 4.8: Update Resources and Metrics for Optimization Task

phases, whereas the *Creation Phase* and *Warm-up Phase* are part of the *Add Optimization Task* use case, described in Section 4.4.4. The *Update Phase* corresponds to the *Update Resources and Metrics* use case.

Creation Phase Parameters that are assigned to values in this phase comprise three categories. First, parameters that belong to the DSP topology graph G_{dsp} are determined by parsing the VISP Topology Description Language file and additionally by requesting the VISP Runtimes via their REST Interfaces. For this, the vertices V_{dsp} are created and their attributes $P_{(CPU,i)}$, $P_{(Mem,i)}$, $P_{(HD,i)}$, $P_{(Cores,i)}$, s_i are assigned. Analogously, the edges E_{dsp} of G_{dsp} are created according to the structure of the DSP topology.

Second, the resource graph G_{res} is created in this phase by applying the Resource Pool mapping process from Section 4.4.3. Some attributes of the resource nodes V_{res} and edges E_{res} are directly derived from the VMs of the corresponding Resource Pools like availability A_u and CPU frequency $P_{(CPU,u)}$. However, the rest of the attributes,

e.g., $P_{(Cores,u)}$ (see Table B.4), need to be computed by considering the relative amount of used capacities of the Resource Pools. For this, e.g., a Resource Pool that has a VM with 3 Cores can be separated into 4 *small* nodes with 0.25 cores, 2 *medium* nodes with 0.5 cores, and 1 *large* node with 1 core.

Third, the optimization preferences are loaded from the VISP Runtimes or, if not available, default-wise from the ODR application properties. Herein, we consider the weights of the objective function (see Equations 4.10 and 4.23) as wells as the cycle period, which is used for dynamic replacements.

Warm-up Phase In this phase, the parameter for the operator execution time per ingested data tuple, i.e., ET_i is assigned. This is done by performing tests with the uploaded DSP topology on the provided Resource Pools as described in Section 4.2.2. The resulting execution times are measured and assigned to ET_i as part of the corresponding node $i \in V_{dsp}$. If the user decides to avoid this phase, then customized values can be set in the ODR application properties.

Update Phase This phase considers all parameters from the previous phases to be updated repeatedly.

4.5 Identified Features and Derived Tasks

We now consider the design, presented in this chapter, as basis for a list of features and work items. The detailed version that is used in development is shown in Table B.1, Table B.2, and Table B.3.

Software Design Setup The design needs to be transformed into an initial code basis. For this, a web application framework has to be set up. Furthermore, packages, interfaces and stubs for key classes are created.

Entity Model The entity classes for the DSP graph, the resource graph, and the optimization preferences as well as the Data Context have to be created. The latter one holds references to instantiated entities.

API The RESTful interface for the ODR Reasoner considers parsing the data for new optimization tasks which comprise the DSP topology, the resource graph, optimization preferences and commands for starting and aborting the dynamic optimization.

Metric Provider and Resource Manager The Metric Provider and the Resource Manager components have to be created for incorporating metrics from the Simulator as well as from the VISP Runtime and to create the internal resource infrastructure. They have to consider the three phases defined in Section 4.4.5.

Scheduler The optimization of placements has to be triggered repeatedly. Therefore, the scheduler component invokes the Data Context, ILP Solver, Persistence Strategy, and Reconfiguration Manager in defined cycles.

Persistence Strategy The number of replacements in each cycle has to be limited. Therefore, heuristics as e.g., migration cost have to be incorporated into the optimization.

ILP Solver The designed optimization model has to be transformed into a suitable specification in Java, that incorporates the parameters from the entity model.

Reconfiguration Manager The optimized placements, received from the ILP Solver, have to be forwarded to a VISP Runtime by uploading this information in a VISP Topology Description Language file.

VISP Client This component has to handle all requests from the ODR Reasoner to the VISP Runtimes. For this, corresponding HTTP requests have to be prepared for retrieving the DSP topology updates, the resource infrastructure, and related metrics (see Section 4.2).

Reporting This component includes a UI that enables visualization of the resource graph and its placed DSP operators. Furthermore, the current QoS metrics (i.e., response time, availability, cost) are shown in a table and exported to a file.

Simulator For testing the placement optimization without involving VISP Runtimes, the Simulator has to provide optimization task requests to be forwarded to the ODR Reasoner API. Furthermore, REST endpoints for the metric updates have to be provided. For this, the delivered metrics are simulated.

Integration To integrate the VISP Runtime with the ODR Reasoner, the exchanged Data Transfer Objects (DTOs) have to be parsed to objects that are stored in the Data Context. The communication protocol has to ensure that the correct REST endpoints are called in the correct phases as described in Section 4.4.5 and in the corresponding order as depicted in Figures 4.3 and 4.4.

CHAPTER 5

Implementation

This chapter considers the identified requirements as well as the presented design of Chapter 4 and discusses different implementation aspects. First, used technologies and the development environment of the ODR Reasoner are discussed. Due to the binding to VISP, we describe the deployment of major components and their relationships. Furthermore, we will focus on the integration of data that is requested from VISP Runtimes. This data needs to be timely incorporated in the system model to solve the optimization problem. After this, we present key parts of the implementation such as the creation, parametrization, and solvation to the ILP model. Finally, we describe a reporting tool that we created for observing the execution of the placement optimization.

5.1 Technologies

The ODR Reasoner is developed as a standalone service with *Java 8*[1] as programming language. In order to make use of REST communication, *Spring Boot*[2] is used as a software framework. Spring Boot further facilitates dependency injection which we use to provide the necessary resources for implemented Java objects. The build lifecycle of the ODR Reasoner is managed with *Apache Maven*[3]. For this, software library dependencies are loaded from a central repository and integrated into the build of the ODR Reasoner. The *Docker-Maven-Plug-in*[4] further supports the deployment process where the ODR Reasoner service is packed into a *Docker Container*[5] and deployed to *Docker Hub*. *Docker Hub* is a registry for *Docker Images* that can be downloaded and instantiated as containers to provide the ODR optimization services.

[1] http://www.oracle.com/technetwork/java/javase/overview/java8-2100321.html
[2] https://projects.spring.io/spring-boot/
[3] https://maven.apache.org/
[4] https://github.com/spotify/docker-maven-plugin
[5] https://www.docker.com/

5.2 Development Resource Infrastructure

To develop and test the ODR Reasoner, we make use of the VISP resource infrastructure as depicted in Figure 5.1. Before invoking the VISP Runtime, tests for different use cases are executed with the Simulator. Both applications are deployed on a local workstation. An *OpenStack*[6]-based private Cloud is used to deploy multiple VMs with VISP resources.

First, we consider the VISP Infrastructure Host VM as deployment location for the VISP Runtime, *RabbitMQ*[7], *MySQL*[8], and *Redis*[9]. All of these resources are executed in containers that are obtained from *Docker Hub*. VISP Runtimes use *RabbitMQ* as the message infrastructure that exchanges data tuples between different operators in a DSP topology. *MySQL* and *Redis* are hosted for storing VISP Runtime data. The second type of VMs that are considered in the Cloud are the VMs of used VISP Resource Pools. For scaling purposes, the amount of Resource Pools VMs can be changed within the OpenStack web interface. Third, a VM hosts the VISP Data Provider that is used to ingest test data tuples into enacted DSP topologies. Considering the data links, visualized as arcs in Figure 5.1, the communication between the local workstation and the Cloud is performed in HTTP. The corresponding endpoints, provided by the VISP Runtime, are discussed in the next section.

5.3 Provided Endpoints and Data

Adding an optimization task for finding placements can be initiated by any VISP Runtime. After that, the ODR Reasoner requests the necessary information like topology, Resource Pools and updated metrics. For this, the VISP Runtime provides various REST endpoints to collect the data for the ongoing optimization process.

Get and Upload Topology Initially the VISP Topology Description Language file of Section 4.4.3 is requested from the VISP Runtime that triggered the optimization. Conversely, after the optimization has finished the updated file (i.e., *concreteLocation* and *size* attributes) is returned. This is realized with a HTTP GET and HTTP POST request respectively. To get information about the operators that are part of the topology, a GET request yields a JavaScript Object Notation (JSON) object as exemplarily given in Listing 5.1. The content which is necessary for building and updating the DSP model starts in line 3 after the name attribute, which identifies the operator. The *frequency* in line 3 refers to $P_{(CPU,i)}$ of operator $i \in V_{dsp}$. The *expectedDuration* in line 4 and *actualDuration* in line 5 are incorporated into the system model as $T_{(max,i)}$ and $T_{(actual,i)}$ respectively. Considering the example values in Listing 5.1, $T_{(actual,i)}$ is not exceeding $T_{(max,i)}$. Therefore, this will not influence any replacement decision of the ODR reasoner.

[6] https://www.openstack.org/
[7] https://www.rabbitmq.com/
[8] https://www.mysql.com/
[9] https://redis.io/

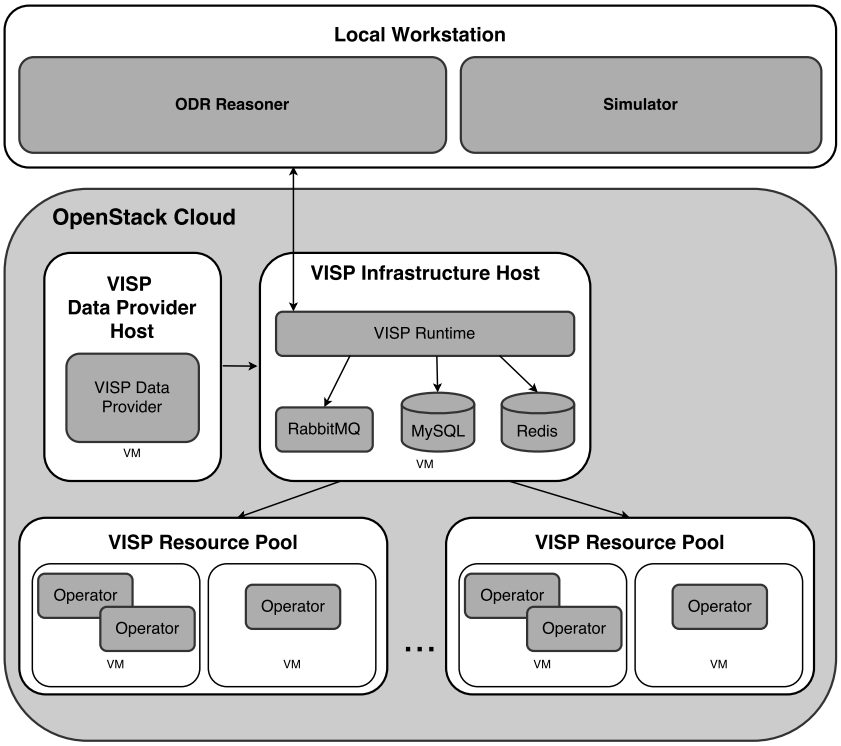


Figure 5.1: Development Environment

Listing 5.1: Operator Configuration

```
1  {
2          "name": "step1",
3          "frequency": 2400,
4          "expectedDuration": 15,
5          "actualDuration": 2,
6          "plannedResources": {
7                  "cores": 0.5,
8                  "memory": 500,
9                  "storage": 300
10         },
11         "actualResources": {
12                 "cores": 0.09,
13                 "memory": 1500,
14                 "storage": 300
15         }
16 }
```

After that, the JSON object distinguishes between *plannedResources* in line 6 and *actualResources* in line 11. Both contain metrics for $P_{(Cores,i)}$, $P_{(Mem,i)}$, and $P_{(HD,i)}$, whereas the *plannedResources* in line 6 are used for initializing the DSP model. The iterative updates of the model are then executed with the values defined in *actualResources*.

In Listing 5.1 we assume that the demand for memory $P_{(Mem,i)}$ increased significantly in contrast to the initial requirements. This increase can lead to a situation where the memory capacity constraint of Equation 4.28 is not satisfied. The ODR Reasoner will therefore try to find an operator placement to provide more memory resources (e.g., deploying the operator on *medium* or *large* sized nodes). Observing the *cores* in line 7 and 12, it can be concluded that in the given example a smaller part of the provided resources is used. So, neglecting the need for more memory, the demand for cores would not lead to place this operator on a larger host. Conversely, a placement at a *small*-sized node could be the consequence.

Get Resource Pools In order to parse the available Resource Pools that can be used for operator placements, we first need to retrieve all available VISP Runtimes. Therefore, the VISP Topology Description Language file is scanned for the VISP Runtime addresses. The Resource Pool information is then requested from each VISP Runtime. A corresponding example result is presented in Listing 5.2. It contains an enumeration of pool names, which act as identifiers for retrieving further details in subsequent requests.

Listing 5.2: Available Resource Pools for a VISP Runtime

```
1 {
2         "largeCloudPool": "pool",
3         "mediumCloudPool": "pool",
4         "externalManufacturingFogPool": "pool"
5 }
```

For this, an exemplary response is shown in Listing 5.3. The *cost* attribute in line 3 refers to C_u, while *cpuFrequency* in line 4 and *availability* in line 5 are assigned to $P_{(CPU,u)}$ and A_u respectively. The response contains multiple objects that describe the provided resources. For ODR purposes, we use *overallResources* stated in line 6 and its *cores* (line 7), *memory* (line 8), and *storage* (line 9) attributes to assign them to $P_{(Cores,u)}$, $P_{(Mem,u)}$, and $P_{(HD,u)}$ respectively. These resources are then partitioned into multiple nodes of different sizes (i.e., *small*, *medium*, *large*) according to the Resource Pool mapping process described in Section 4.4.3.

Listing 5.3: Resource Pool Information

```
1  {
2          "name": "largeCloudPool",
3          "cost": 20.5,
4          "cpuFrequency": 2400,
5          "availability": 0.995,
6          "overallResources": {
7                  "cores": 8,
8                  "memory": 15360,
9                  "storage": 40960
10         }
11         ...
12 }
```

Get Data Link Attributes Considering multiple decentralized VISP Runtimes, the network infrastructure shows different conditions. Especially in a Fog environment, the data link attributes vary. In order to retrieve this information, VISP Runtimes, which provide their services for hosting DSP operators, are queried. To describe the received responses, we consider an example in Listing 5.4.

Listing 5.4: Data Links between VISP Runtimes

```
1  [
2          {
3                  "start": "128.120.172.182",
4                  "end": "128.120.172.182",
5                  "delay": 0.005,
6                  "dataRate": 3193,
7                  "availability": 0.995
8          },
9          {
10                 "start": "128.120.172.182",
11                 "end": "128.120.0.1",
12                 "delay": 0.077,
13                 "dataRate": 3193,
14                 "availability": 0.9875
15         }
16 ]
```

The first entry describes a self-link of the requested VISP Runtime. This link connects a VISP Runtime with itself, since its *start* and *end* addresses are equal. However, the second entry refers to a link between two different VISP Runtimes, that manage different Resource Pools. The information in the properties *delay* (line 12) and *availability* (line 14) are assigned to $d_{(u,v)}$ and $A_{(u,v)}$ respectively. So, if node $u \in V_{res}$ is managed by VISP Runtime R_A and node $v \in V_{res}$ by R_B, then the edge (u, v) is assigned with values defined in the JSON entry where the addresses of R_A and R_B are used for *start* and *end* respectively. In order to compute $b_{(M,u)}$, the average of all scanned *dataRate* attributes in line 13 is considered.

5.4 Optimization

After the parameters have been incorporated into the resource and DSP model, the optimization can be started to solve the placement problem. For this, the Java API of the *IBM CPLEX Optimizer*[10] is used. It enables to create the ILP optimization model with its objective function and several constraints. Herein, we update parts of the CPLEX specific Java code from Cardellini et al. [10].

Listing 5.5 presents an example of adding a constraint that refers to Equation 4.28. *IloCplex* is the main object that collects all modeled expressions. For the creation of these, the *IloModeler* is used. It constructs *IloLinearNumExpr* objects like *memCapacity*

[10] https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

(line 3), which consists of a sum of terms. A term refers to a product of a numeric value and an *IloNumVar* as returned by *X.get(i,u)* in line 9. The specially created *PlacementX* *X* object is considered as a collection of decision variables (e.g., *IloNumVar*), referring to $x_{i,u}$ in the system model.

Listing 5.5: CPLEX Constraints in JAVA

```
1  void addMemoryConstraint(PlacementX X, IloCplex cplex, IloModeler modeler) {
2          for (ResourceVertex uRes : this.resGraph.getVertices().values()) {
3                  IloLinearNumExpr memCapacity = modeler.linearNumExpr();
4                  for (DspVertex iDsp : this.dspGraph.getVertices().values()) {
5                          int i = iDsp.getIndex();
6                          int u = uRes.getIndex();
7                          if (iDsp.deployableOn(u)){
8                                  memCapacity.addTerm(iDsp.getRequiredMem(),
9                                                          X.get(i, u));
10                         }
11                 }
12                 cplex.addLe(memCapacity, uRes.getAvailableMem(),
13                                 "cap_mem_res_" + uRes.getIndex());
14         }
15 }
```

In line 12 the call *cplex.addLe(...)* appends the constraint to the existing model by considering '$\leq$' as inequality operator. As it can be seen in Listing 5.5, the entities like *ResourceGraph*, *ResourceVertex*, *DspGraph* and *DspVertex* are used for storing the parameters of the system model (see Section 4.2). To add the corresponding parameters to CPLEX, we iterate over this data structure to create terms for the memory capacity constraint. Similarly, this also holds for constructing other constraints as well as the objective function in CPLEX. If finally the model is completely created, the *IloCplex* object is used to compile and solve the ILP placement problem.

We consider the solution of the operator placement problem with CPLEX in Listing 5.6. The method *solve()* uses the created optimization model (see Section 4.3) with the relevant objective function and constraints in the global *cplex* object and applies a simplex algorithm to solve the linear program.

First, we consider two configuration possibilities that can ease the solution procedure. In line 4 of Listing 5.6, a time limit for the optimization duration is set. If the optimization would last longer than the defined time limit, CPLEX stops the optimization and returns the best solution computed so far. Similarly, we consider an allowed optimality gap in line 7. CPLEX considers the optimality gap as a scalar distance from the best value of the objective function that can be obtained. So, if CPLEX computes a solution that has an objective function value which deviates less than the defined optimality gap, then CPLEX stops. The time- and optimality gap-based stopping criteria can be used, especially when the problem space, e.g., resource graph, increases in size.

The actual solution procedure is started in line 10. In line 14 we instantiate the object that contains the results like operator placements and QoS attributes, that are extracted

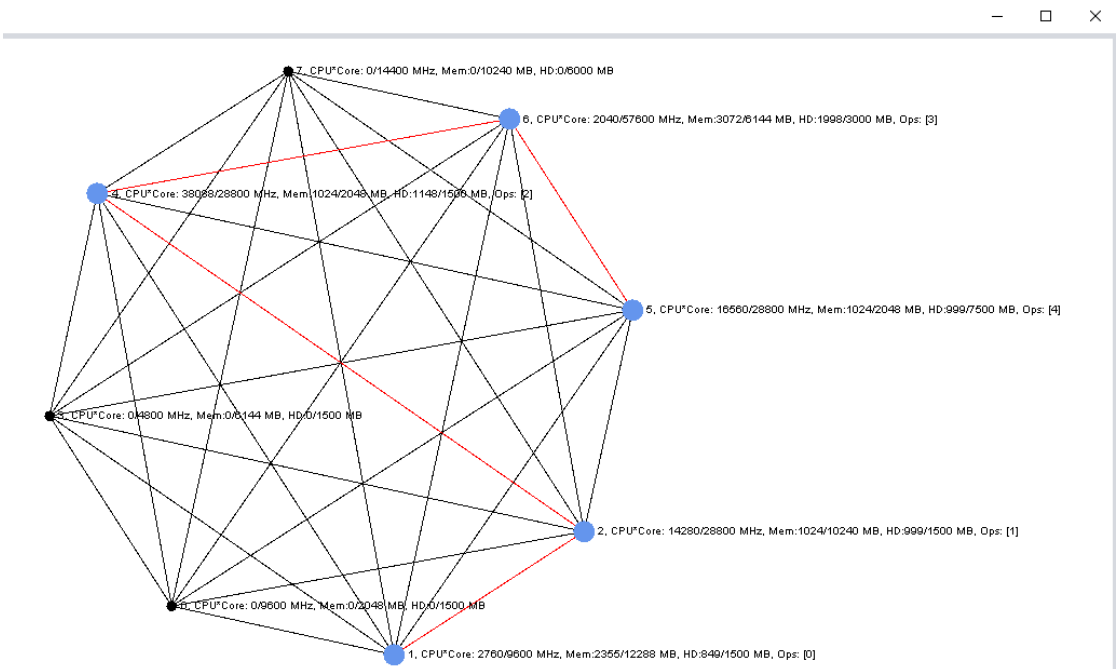
and added to the solution object in lines 15-19. For this, with *cplex.getValue(...)* we can retrieve the values of the terms that have been defined similarly to the *memCapacity* object of class *IloLinearNumExpr* as shown in Listing 5.5. Herein, the terms response time R, logarithmic availability *logA*, enactment cost *enactC*, and migration cost *migC* are the QoS attributes for the optimized operator placements (see Equations 4.10 and 4.23). Finally, we extract every placement of a DSP operator with index i on the assigned resource node with index u. For this, we use the call *cplex.getValue(X.get(i,u))* to retrieve the corresponding placement status of decision variable $x_{i,u}$ from CPLEX in line 28. The *solution* object is then used in subsequently invoked components for reporting the result and uploading the placement information to a VISP Runtime.

Listing 5.6: Solve ILP Problems with CPLEX in JAVA

```
1  OptimalCplexSolution solve() {
2          /* limit the solution time if required */
3          if (defaultParams.usesTimeLimit()){
4                  cplex.setParam(IloCplex.IntParam.TimeLimit, defaultParams.getTimeLimit());
5          }
6          if (defaultParams.isOptimalityGapDefined()){
7                  cplex.setParam(IloCplex.DoubleParam.EpGap, defaultParams.getGap());
8          }
9
10         /* apply ILP solving by making use of the simplex algorithm */
11         cplex.solve();
12
13         /* extract resulting QoS attributes and store it in the solution object */
14         OptimalSolution solution = new OptimalCplexSolution(dspGraph.getVertices().size());
15         solution.setOptObjValue(cplex.getObjValue());
16         solution.setOptR(cplex.getValue(R));
17         solution.setOptLogA(cplex.getValue(logA));
18         solution.setOptEnactmentCost(cplex.getValue(enactmentC));
19         solution.setOptMigCost(cplex.getValue(migC));
20
21         /* extract placements */
22         for (DspVertex dspOperator : dspGraph.getVertices().values()) {
23                 for (ResourceVertex resNode : resGraph.getVertices().values()) {
24                         int i = dspOperator.getIndex();
25                         int u = resNode.getIndex();
26                         double xval = 0;
27                         if (dspOperator.deployableOn(u))
28                                 xval = cplex.getValue(X.get(i, u));
29                         if (xval > 0)
30                                 solution.setPlacement(i, u);
31                 }
32         }
33         return solution;
34 }
```

5.5 Reporting

In order to observe the placement optimization, the ODR Reasoner contains a *Reporting* component as presented in the static view of the software design in Section 4.4.1. Throughout optimization, the topology operators are visualized together with the resource graph, where they are deployed on. Figure 5.2 depicts these graphs, whereas the blue-marked nodes are resource nodes with placed operators. Labels are used to visualize the current resource utilization with respect to memory, storage, and the product of CPU frequency and cores. The table lists the current values of QoS attributes. This data is part of a spreadsheet, which is exported in each optimization cycle. We consider this to evaluate the impact of ODR Reasoner against baseline approaches as presented in the following Chapter 6.



Resource Nodes	Topology Nodes	Response Time (s)	Availability	Provisioning Cost (€ / s)	Total Migration Cost (€ / iteration)
8	5	86,00	74,11 %	5,00	0,00
8	5	89,23	73,66 %	5,00	1,77
8	5	89,23	73,66 %	5,00	1,77
8	5	89,23	73,66 %	5,00	1,77

Figure 5.2: Placement Optimization Monitoring

CHAPTER 6

Evaluation

To show how the implemented ODR Reasoner performs, we present different evaluation scenarios. Therefore, the following chapter describes general prerequisites like used test beds, optimization parameters, DSP topologies and ingested data that is processed by the operators. Then, we present each scenario with its specific setup, evaluation hypothesis, and the resulting QoS metrics that are measured or computed throughout the optimization. For each scenario, we discuss their execution and results and observations. Finally, a summary of the overall outcome concludes this chapter.

6.1 Prerequisites

6.1.1 Test Beds

Before the evaluation can be started, the resource infrastructure has to be configured. Like in the development phase, we make use of a private OpenStack Cloud, located at TU Wien. According to a given scenario, as described below, VMs are used to serve as VISP Infrastructure Hosts and to form multiple corresponding Resource Pools (see Figure 5.1). Due to multiple scenarios, we create multiple test beds which are suitable for a pure Cloud-based scenario as well as a simulated Fog-based scenario. For the Fog Computing scenarios, we manipulate the network delay between selected VMs with the tool *tc*[1] that allows network traffic shaping (e.g., slow down certain connections). In the evaluation phase, the ODR Reasoner is deployed to a local workstation, which allows the observation of the optimization process by making use of the created reporting tool as presented in 5.5.

[1] `https://linux.die.net/man/8/tc`

6.1.2 Topologies

All presented scenarios consider one base topology whose attributes are adapted for the given resource infrastructure used in the respective scenario. Figure 6.1 depicts this topology consisting of four operators and a source as well as a sink. Depending on the resource infrastructure for a given scenario, the VISP Topology Description Language attribute *allowedLocations* is set accordingly. For this, we define which Resource Pools are considered as deployment locations. Regardless of the scenario, each operator starts with a *size* attribute *small*, since scaling to *medium* or *large* is performed during optimization. The processing operation for each operator is defined by waiting for a fixed time interval and then forwarding the tuple to the next operator.

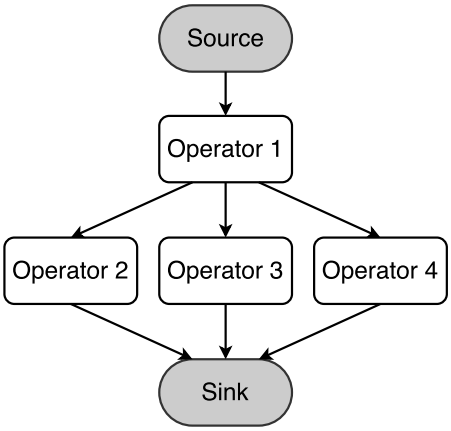


Figure 6.1: Topology for Evaluation Scenarios

6.2 Mapping of IoT Manufacturing Scenario

In Section 1.1.1 we introduced a topology of the motivational scenario (IoT manufacturing) as shown in Figure 1.1. To show that the results of the evaluation hold true for the motivational scenario, we provide a mapping between the two topologies and resource infrastructures.

Topology Mapping The motivational topology contains *Availability*, *Productivity*, and *Temperature* sensors that we generalize to one sensor given as *Source* in the evaluation topology (see Section 6.1.2). The operators *01 Distribute Data*, *O2 Filter Availability*, and *O6 Monitor Temperature* receive data from their input sensors and apply the respective processing steps. We abstract these operators and map them to *Operator 1* of the evaluation topology. *O7 Calculate Overall Equipment Effectiveness* and *O8 Maintenance Call Service* are mapped to the successors, i.e., *Operator 2* and *Operator 3* respectively, of *Operator 1* in the evaluation topology. Finally, *O9 Dashboard Service* consumes the data from its predecessors and execute the last step in the motivational topology. Therefore, we map this Operator to the *Sink* in the evaluation topology.

Resource Infrastructure Mapping As it is the case in the Fog Computing evaluation scenario (see Section 6.3), the motivational scenario contains two Resource Pools in the Cloud. Therefore, we map the *Public Cloud* and *Private Cloud* from the motivational scenario to the respective *Cloud* Resource Pools. The resources that are available on the manufacturing site, i.e., *Cloudlet (Smart Factory 1)* and *Cloudlet (Smart Factory 2)* in the motivational scenario corresponds to *Fog Node 1* and *Fog Node 2* in the evaluation scenario. The discussed Cloudlets consider a high latency to the *Public Cloud* and *Private Cloud*, which behave as in the evaluation scenario, whereas 400 ms are assigned as latency between *Cloud* and the *Fog* nodes.

6.2.1 Data Ingestion & Forwarding

The VISP Data Provider is configured to ingest data into the evaluated topology. For this, it generates on average one message per second. Nevertheless, the message generation frequency is not constant. For evaluation purposes, we select a generation pattern in the form of a sinus function to simulate different load scenarios. In the topology presented in Section 6.1.2, the operators consider processing times as shown in Table 6.1. Therefore, each operator waits before the tuples are forwarded to the succeeding operator.

Table 6.1: Processing Times of Operators

Operator	Processing Time
1	100 ms
2	250 ms
3	500 ms
4	1000 ms

6.2.2 Optimization Preferences

In contrast to ODP [10], the presented ODR approach considers dynamic replacements of operators. Therefore, we consider an optimization cycle period of 4 minutes after which the ODR Reasoner evaluates the system and potentially suggests operator replacements. The overall evaluation for a scenario lasts for 50 minutes. Furthermore, the scenario is executed three times, whereas the resulted QoS metrics are averaged to provide more robust results. This is necessary since we want to avoid results that are biased by potential monitoring errors or delays in replacement activities caused by the test environment. As mentioned in the design, we consider every *small* node, which is created during the Resource Pool Mapping (see Section 4.4.3), to have a capacity of 2400 Mhz, 0.2 cores, 512 MB memory, and 300 MB storage, based on the available VMs in the Resource Pool. The execution time per data tuple ET_i is set to a fixed value for each operator $i \in V_{dsp}$, due to the fixed processing time in the operators.

6.2.3 Baseline Approach

The baseline for every scenario, which is used for comparing with ODR, is a static optimization approach. This refers to only optimizing placements at system startup and avoiding any replacement at run time. For comparison reasons, the resulting QoS metrics are recorded for the whole evaluation by requesting the relevant input metrics from the VISP Runtimes. So, although the optimization is only applied once in the static baseline approach, metric updates are still performed in a cyclic manner (see Section 4.4.4). The rest of the optimization preferences remain the same between the dynamic and static approach.

6.2.4 Evaluation Tools

To set up the testbeds and to execute the evaluation scenarios, we developed an *Evaluation Suite*. It is a Java program that controls the preparation and execution of all scenarios. It invokes *Selenium*[2] and a Java API, named *sshj*[3] for sending Secure Shell (SSH) commands. The former one allows to automatically control web browsers to test web applications. We used *Selenium* for respective VISP Runtime web interfaces for automating the resource pool creation, uploading of topology descriptions, triggering optimization tasks, and performing a clean-up after the evaluation has finished. The *SSH* commands are used to reboot all involved services and applications by restarting the corresponding container installed on the VISP Infrastructure Hosts. This is done before each new evaluation scenario run to have a clean environment. As discussed above, SSH is also used for executing *tc* network traffic shaping commands.

6.2.5 Resource Pool VMs

To fill the Resource Pools with deployed operators, the OpenStack VMs have to be provided. For this, we consider three OpenStack VM sizes, named *flavors*, in the evaluation scenarios as described in Table 6.2. The VMs are created by the *Evaluation Suite* before uploading the topology and starting the optimization task. Besides hosting Resource Pools, we also run the VISP Infrastructure, i.e., Redis, MySQL, and RabbitMQ, on VMs. By default, the *flavor* for this is *m1.medium*.

Table 6.2: OpenStack Pool VM Flavours

Flavor	Memory (GB)	VCPUs	Storage (GB)
m1.medium	3	2	40
m2.medium	5	3	40
m1.xlarge	15	8	40

[2] `http://www.seleniumhq.org/`
[3] `https://github.com/hierynomus/sshj`

6.3 Fog Computing Scenarios

In the evaluation of the ODR Reasoner we distinguish between two categories of scenarios. The first category considers a simulated Fog infrastructure that is described in this section. The second category focuses on a Cloud infrastructure as discussed in 6.4.

6.3.1 Test Bed Details

Resource Infrastructure The resource infrastructure that is used in the Fog Computing scenarios is depicted in Figure 6.2. We set up three VISP Runtimes, whereas the *Cloud* Runtime manages two resource pools. To simulate an extra Fog environment, we created dedicated VISP Runtimes for *Fog Node 1* and *Fog Node 2*. The Fog scenarios start with two VISP Runtimes (*Cloud* and *Fog Node 1*), since *Fog Node 2* is not available at the beginning of the optimization.

To motivate this, we assume a department of an enterprise that uses Resource Pool VMs with different cost components described in the *Resource Pool Cost Model* in Table 6.3. We distinguish between depreciation, maintenance and leasing cost components that add up to the total cost per second. Depreciation is the process of allocating cost of an asset (e.g., hardware) over its expected life time [75]. To illustrate this, we assume e.g., hardware acquisition cost of 1000 Currency Units (CUs) for a life time of 5 years. For this, the depreciation per year results in $1000/5 = 200$ CU. *Maintenance* cost refers to hardware and software maintenance of the given infrastructure. Leasing cost includes a fee that has to be paid to the owner of the used resources.

The Cloud VMs in the Fog Computing scenario have to be leased with cost of 15.5 CU/s for *m1.medium* and 20.5 CU/s for *m2.medium* as shown in Table 6.3. Since there is no hardware acquisition, we do not consider depreciation and maintenance cost. *Fog Node 1* provides computing resources at a high price (30.5 CU/s), since the maintenance for this on-premises infrastructure is expensive (29.5 CU/s). The high maintenance cost is on the one hand due to the difficult hardware maintainability of *Fog Node 1* which is located at the manufacturing site of the enterprise. On the other hand, the resource is not shared with any other department. So, the cost centre of the department has to bear all the cost of *Fog Node 1*. Nevertheless, this resource has to be used because it hosts the *Operator 1* of the topology to pre-process the data received from the *source* sensor. *Fog Node 2* is located on the manufacturing site too but its computing resources are shared between multiple departments and its installed hardware and software can be maintained easily. Therefore, we consider very low maintenance cost of 1.5 CU/s which adds up together with the depreciation of 1 CU/s to 2.5 CU/s in total. To explain why *Fog Node 2* is not available at the beginning of the optimization, we consider it to be occupied for the first 20 minutes of the DSP evaluation period by other departments. After that, it serves as a potential host for DSP operators. Besides the Resource Pools, we consider an *Optimization Server* located in the department office. This server is in charge of executing the placement optimization at a total cost of 3 CU/s.

Table 6.3: Resource Pool Cost Model

Resource Pool	Depreciation (CU/s)	Maintenance (CU/s)	Leasing (CU/s)	Total (CU/s)
Cloud (m1.medium)	0	0	15.5	15.5
Cloud (m2.medium)	0	0	20.5	20.5
Fog Node 1	1	29.5	0	30.5
Fog Node 2	1.5	1	0	2.5

The behavior of a Fog device that is not available at a certain spot in the network (e.g., near the manufacturing site) and then suddenly appears to provide its resources, is a central characteristic of Fog Computing. This is referred to as *Mobility* [3]. Although in our case the *Fog Node 2* is not moved, but rather used by other parties, it can be seen as a matter of limited localized resources [76]. Unlike in the Cloud, where the amount of resources are practically infinite, Fog Resources in specified regional areas can be scarce. So, these resources have to be shared with other parties in an alternating way. This leads to situations, where Fog users have to wait until the requested resource is available for executing their services. In our case, as soon as *Fog Node 2* is available, it is registered as Resource Pool to be managed by the *VISP Runtime Fog Node 2*. For evaluation purposes, the registration is performed by the *Evaluation Suite*.

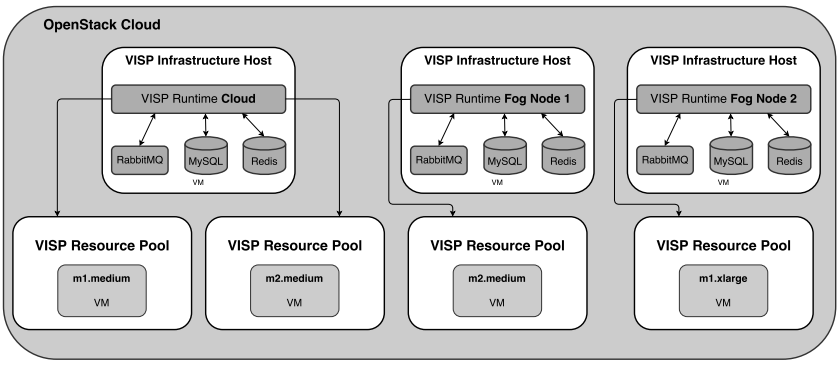


Figure 6.2: Fog Simulation Resource Infrastructure

Topology Specification The topology used in the Fog scenario is defined with the VISP Topology Description Language as shown in Listing A.1. Three allowed locations (i.e., VISP Runtimes) are defined as e.g., in line 20, whereas the *Cloud* Runtime has the address *128.130.172.182* assigned and *Fog Node 1* and *Fog Node 2* refer to *128.130.172.183* and *128.130.172.217* respectively. The defined concrete locations are chosen to be equally distributed over the used resources. *Operator 1* is fixed on the Resource Pool *fogPool1* of *Fog Node 1*.

Network Traffic Shaping To simulate a Fog environment, we manipulate the network delay by making use of *tc*. In this scenario category, we consider no network delay for the

resources hosted on the Resource Pools of the *Cloud* Runtime. In contrast, the Resource Pool of *Fog Node 1* faces a delay of 400 ms, due to its remote location. When *Fog Node 2* is available after 20 minutes, we simulate that this node is very close to *Fog Node 1*. For this, we set the network delay between *Fog Node 1* and *Fog Node 2* to 10 ms. The latency between *Fog Node 2* and the *Cloud* is 400 ms. This leads to the situation that the latency between the *Cloud* and the two Fog nodes on the manufacturing site is relatively high.

6.3.2 Full Model

In the Fog Computing scenario category, we start to evaluate the *Full Model* approach of the ODR Reasoner. *Full Model* refers to the ILP optimization model that considers all QoS attributes in the objective function (see Section 4.3.2) with equal importance, i.e., $w_r = w_a = w_{c_{op}} = w_{c_{mig}} = 0.25$.

Evaluation Hypothesis The first hypothesis addresses the performance with respect to the QoS attributes, while the second hypothesis focuses on the cost:

> *H1: In a Fog environment, the dynamic ODR approach is better than the static approach with respect to (a) response time and (b) availability.*

> *H2: In a Fog environment, the sum of all cost of the dynamic ODR approach is less than the cost of the static approach.*

In general, we aim to analyze the impact of ongoing optimization. Considering *H1*, we know that there can be a trade-off between the identified QoS attributes. Nevertheless, the ODR reasoner can incorporate dynamic changes in the environment to improve the results for both attributes. So, if new Fog Resources are detected, there is a chance to increase response time and availability. Considering *H2*, the introduced migration cost has to be compensated by decreasing the enactment cost.

Evaluation Results To show the results of this evaluation run, we distinguish between response time, availability, and cost metrics. Furthermore, we discuss the total score of the dynamic optimization and compare it with the corresponding static optimization result. The total score is considered as the maximized value of the objective function (see Equation 4.10 and Equation 4.23). CPLEX returns this value after the solution has been found.

Starting with Figure 6.3, the response time is plotted on a chart with the time progress of the evaluation displayed on the x-axis and the actual response time of the deployed DSP topology on the y-axis. For the optimization duration of 50 minutes, the recordings of the dynamic approach are visualized with the solid line, whereas the dashed line belongs to the static approach. The red line, which is vertically drawn at the 20 minutes mark, represents the time point when *Fog Node 2* is ready for providing resources to

host operators. For this, *Fog Node 2* is registered as *m1.xlarge* Resource Pool on the corresponding *VISP Runtime Fog Node 2*.

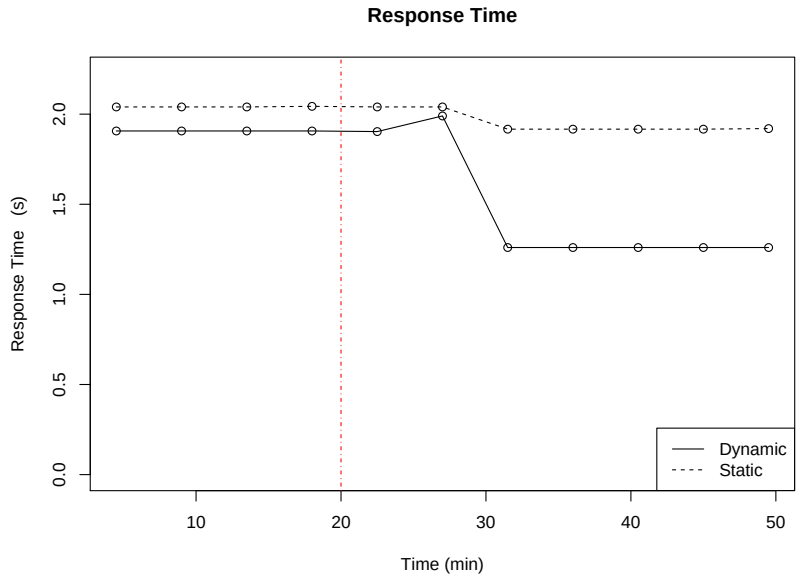


Figure 6.3: Response Time in the Full Model Fog Scenario

The response time of the dynamic approach is permanently lower than the response time of the static approach. Interestingly, at the beginning of the optimization, the dynamic line starts below. One reason can be that minor deviations in the monitoring data are present at given points in time. This can lead to changes in the boundary parameters like R_{max}, R_{min} (see Section 4.4.4), which in turn changes the objective function (see Section 4.3.2). Hence, the static ODR approach achieves a higher total score by focusing on other terms of the objective function (e.g., cost, availability). After that, the resulting placements can be e.g., more cost efficient but cause longer response times. However, after *Fog Node 2* is available, the dynamic line temporary approaches the static line until the ODR Reasoner performs replacements that significantly decrease the response time to 1.3 seconds. Compared to 1.9 seconds of the static approach, the improvement achieved by the dynamic ODR optimization amounts to 31.5%. The ODR Reasoner exploits the advantages of a lower network delay (i.e. 10 ms) between *Fog Node 1*, where the *Operator 1* is deployed, and *Fog Node 2*. For this, the high latency (400 ms) between the two Fog nodes and the Cloud is mostly avoided by moving the operators to the Fog (i.e., *Fog Node 1* and *Fog Node 2*).

In Figure 6.4 the availability of the overall topology, as defined in Equation 4.5, is visualized. Again, the dynamic and static approaches are compared over time which is displayed on the x-axis. The availability is plotted on the y-axis. It should be noted that due to a missing availability monitoring component, the received availability metrics

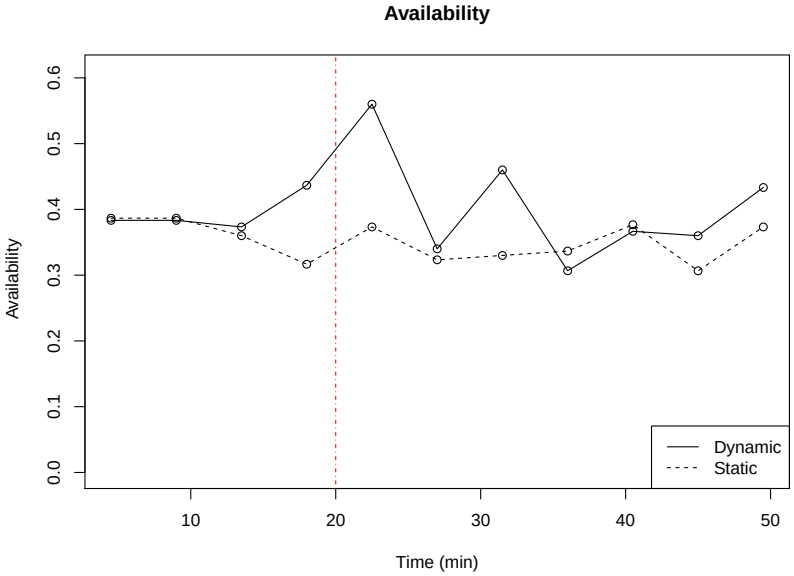


Figure 6.4: Availability in the Full Model Fog Scenario

A_u and $A_{(u,v)}$, are generated randomly every time when they are requested from the VISP Runtime. The metrics follow a uniform distribution with 0.8 and 1 as boundaries, i.e., $A_u \sim U(0.8, 1)$ and $A_{(u,v)} \sim U(0.8, 1)$. Therefore, the availability scale contains a broad range of values. Nevertheless, to reduce the impact of randomization, we consider multiple evaluation runs, whereas its results are averaged as stated in 6.2.2.

Focusing on the results, we see that multiple peaks are reached by the dynamic approach. In general the availability fluctuates more than it is the case for the static approach. This is due to the additional amount of placement possibilities that can be exploited by the dynamic approach, while the static approach cannot react to the resources offered by *Fog Node 2* after 20 minutes. In contrast to the response time perspective, as presented in Figure 6.3, the dynamic approach provides a better availability in only 9 out of 11 optimization cycles, whereas in 2 points the static approach is marginally better. One reason for that is the structure of the objective function (see Equation 4.10 and Equation 4.23), which considers multiple QoS attributes that are equally weighted in the current scenario. Hence, the placement decision at a given point in time cannot be made just to optimize one specific QoS attribute as e.g., availability. Overall, the availability of DSP topologies, hosted in a Fog environment, can be improved if dynamic placement optimization is applied.

Figure 6.5 depicts the cost of the enactment over time. Herein, we consider three lines that are plotted. First, the dashed line refers to the total cost of the static approach, which basically consists of the enactment cost (see Equation 4.19) and initial optimization cost. Second, the dot-dash line refers to the enactment cost of the dynamic approach.

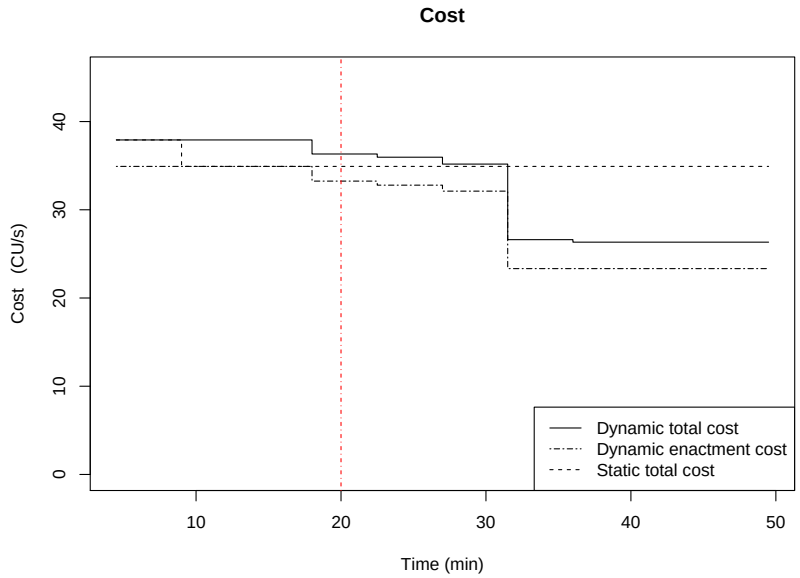


Figure 6.5: Cost in the Full Model Fog Scenario

Third, the total cost of the dynamic approach as a sum of the enactment, migration cost, and the optimization cost, i.e., cost for the *Optimization Server*, is plotted as a solid line. The cost on the y-axis is measured as CU/s. Therefore, we can keep the enactment cost unmodified, but have to compute the migration cost per second, as they are measured per optimization period (i.e., 4 minutes). After that, we finally add up the enactment, optimization, and migration cost.

The first observation is that the enactment cost of the dynamic approach only differs slightly from the respective total cost. This is because the migration cost is very low compared to the enactment cost. So, the main difference between the two lines is caused by the fixed optimization cost, that has to be paid continuously. In contrast, the optimization cost in the static approach only accrues at the first optimization cycle. Afterwards the optimization cost decreases to 0, since the respective resource is released. Comparing the total cost of both approaches, we see that the dynamic approach has the disadvantage of the optimization cost that needs to be compensated with cost-efficient placement decisions. We see that the solid line of the dynamic approach decreases significantly when *Fog Node 2* is available. Nevertheless, right before the 20 minutes mark, the dynamic approach found a placement that costs less than the initially optimized placement. Hence, we conclude that not only structural changes, introduced with *Fog Node 2*, but also changes in e.g., monitoring data lead to replacements of operators that improve the performance of QoS attributes. Overall, the total cost savings that are achieved by the dynamic approach amount to 3.09 CU/s, i.e., 8.8%, with respect to the static total cost as shown in Table 6.4.

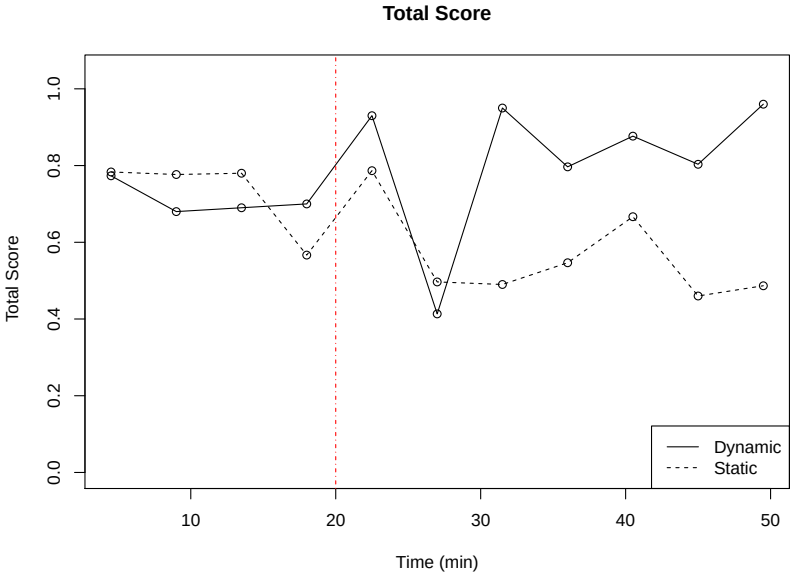


Figure 6.6: Total Score in the Full Model Fog Scenario

Figure 6.6 depicts the total score, i.e., maximized value of the objective function. After a fluctuating start until the 30 minutes mark, the advantages originated from *Fog Node 2* can be observed. The total score of the dynamic line shows better results until the end of the evaluation period.

To show how ODR optimizes operator replacements, Figure 6.7 depicts an example replacement activity that is executed in one of the three evaluation runs in this scenario right after *Fog Node 2* is available. Before the replacement, *Operators 2-4* are placed at *cloudPool1*, whereas *Operator 1* is fixed at *fogPool1* where the *Source* is located. The *Sink* is placed at *cloudPool2* as specified in the topology (see Section 6.3.1). The replacement action migrates the *Operators 2-4* to *Fog Node 2* to exploit the advantages in response time and cost. This leads to improvements in both criteria as presented in Figure 6.3 and Figure 6.5 respectively.

To summarize the results of this scenario, we consider Table 6.4 that shows the means and standard deviations of the used QoS attributes and reported key figures with respect to three evaluation runs. We can see, that on average the dynamic approach is better than the static baseline with respect to response time, availability, enactment cost, total cost, and the total score.

To verify the hypothesis *H1* and *H2* we sum up the results. Part (a) of hypothesis *H1* can be shown with respect to the response time, since the dynamic optimization approach of the ODR Reasoner always provides shorter response times than the corresponding static approach. The availability of the dynamic line in Figure 6.4 is fluctuating, but in

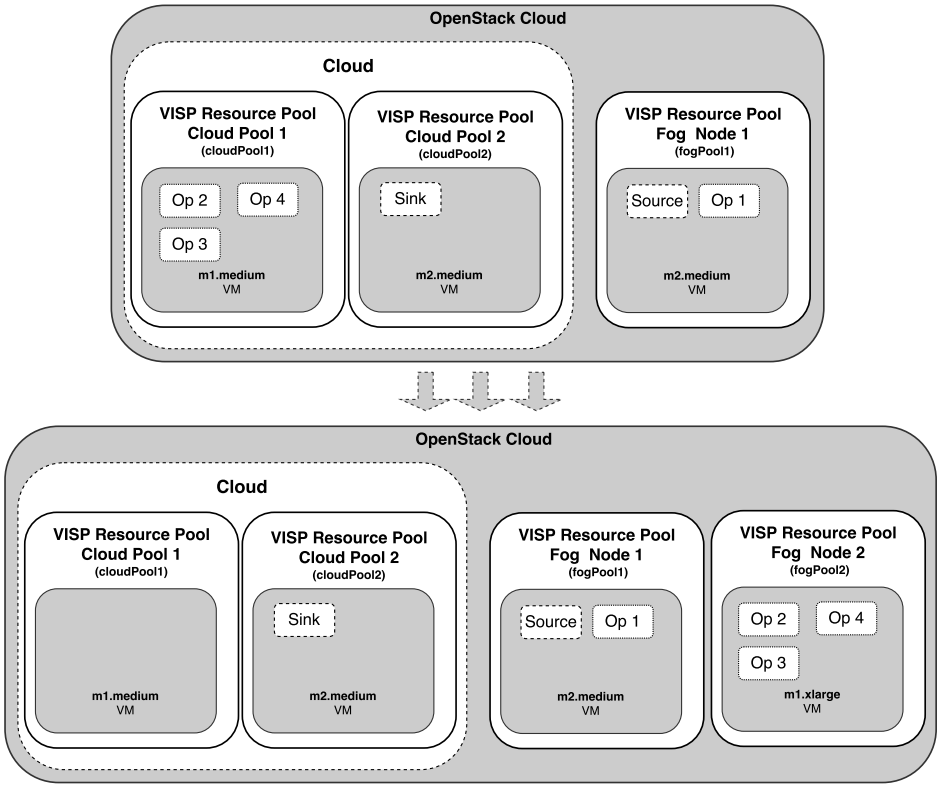


Figure 6.7: Example Replacement Activity

Table 6.4: Result Metrics for the Full Model Fog Scenario

	Static	Dynamic
Average Response Time (sec)	1.98 ($\sigma = 0.12$)	1.62 ($\sigma = 0.01$)
Average Availability	0.35 ($\sigma = 0.07$)	0.40 ($\sigma = 0.06$)
Enactment Cost (CU/s)	34.91 ($\sigma = 2.09$)	29.05 ($\sigma = 0.65$)
Migration Cost (CU/s)	-	14.47 ($\sigma = 1.17$)
Optimization Cost (CU/s)	0.27 ($\sigma = 0$)	3 ($\sigma = 0$)
Total Cost (CU/s)	35.19 ($\sigma = 2.09$)	32.10 ($\sigma = 0.65$)
Cost Savings (CU/s)	-	3.09 ($\sigma = 2.65$)
Average Total Score	0.62 ($\sigma = 0.05$)	0.78 ($\sigma = 0.04$)

the majority of the measurements it is higher than the availability of the static approach. Nevertheless, this is not enough to satisfy part (b) of *H1*.

H2 can be considered as valid by regarding the comparison of total cost between the dynamic and the static approach in Figure 6.5. We can observe that although migrations are performed and their cost has to be considered, the placements in the dynamic approach cost less than the ones of the static approach for the overall evaluation period.

6.3.3 Response Time Model

The *Response Time Model* differs from the *Full Model* with respect to its weights. Instead of choosing equal importance for all QoS attributes in the objective function (see Equation 4.23) by assigning equal weights, we only focus on the response time, i.e., $w_r = 1$ and $w_a = w_{c_{op}} = w_{c_{mig}} = 0$. Therefore, we neglect the availability, enactment cost, and migration cost components of the objective function. However, the constraints of the optimization problem remain unchanged.

Evaluation Hypothesis In this scenario, the *Response Time Model* is used to evaluate the third hypothesis that is formulated as follows:

> *H3: In a Fog environment, the dynamic ODR approach that optimizes solely the response time is better than the corresponding static approach.*

In contrast to the *Full Time Model* scenario, whereas the ODR Reasoner has to consider the trade-off between the involved cost-, response time- and availability-oriented QoS attributes, the total score in the *Response Time Model* directly reflects the response time performance.

Evaluation Results The results of this scenario are depicted in Figure 6.8. The response time on the y-axis is recorded over time i.e., 50 minutes which can be seen on the x-axis. The solid line of the dynamic approach is constantly on the same level (i.e., 1.9 sec) as the dashed line of the static approach for the first 5 optimization cycles. Herein, one cycle lasts approximately 4 minutes in general but due to additional time for solving the placement problem it sums up to 4.5 minutes approximately. However, after the 20 minutes mark, *Fog Node 2* is available and the dynamic line decreases over 3 cycles in a row until it reaches a response time of 1.26 seconds. This yields an improvement of 0.64 seconds, i.e., 33.5%, at the end of the evaluation period. We conclude, that the convergence to a placement with minimum response time may not be achieved immediately. That is, because the ODR Reasoner needs to settle and re-evaluate the situation in the network before further replacement actions are executed. Nevertheless, the optimization is always executed with the goal of achieving a global optimum but as Figure 6.8 depicted, new optimization possibilities exist after receiving feedback via the VISP Runtime monitoring activities. For this, multiple change triggers are re-evaluated as e.g., the actual process duration of an operator $T_{(actual,i)}$ described in Equation 4.26 or network delay $d_{(u,v)}$. In Table 6.5, we compare the mean and standard deviation of the average response time. For this, the dynamic approach dominates clearly. Nevertheless, the average response time statistics only show an improvement of 0.25 sec, i.e., 13%. We conclude that the overall optimization duration needs to be long enough to exploit even more changes in the network and to further reduce the response time.

To verify hypothesis *H3*, we point out that there is no situation where optimized placements of the dynamic ODR approach perform worse than the placements fixed initially

Table 6.5: Result Metrics for the Response Time Model Fog Scenario

	Static	Dynamic
Average Response Time	1.98 ($\sigma = 0.15$)	1.73 ($\sigma = 0.01$)

by the static approach. The advantage becomes clear when the structure of the network changes after 20 minutes, which enables replacements of operators. Hence, *H3* is assumed to be valid.

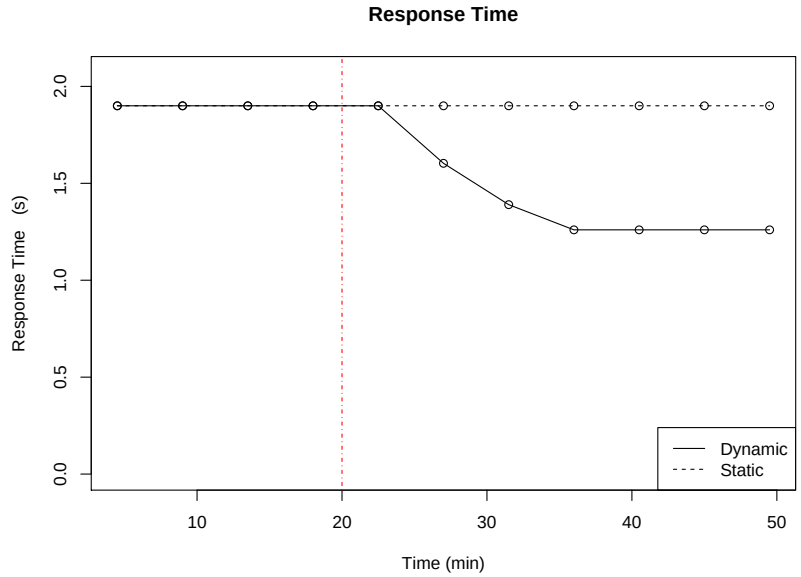


Figure 6.8: Response Time in the Response Time Model Fog Scenario

6.4 Cloud Computing Scenario

To show how the ODR Reasoner performs on a Cloud Computing environment, we now present the evaluation on a Cloud test bed. The changed characteristics are highlighted and the minor changes in the topology are explained in the following.

6.4.1 Test Bed Details

Resource Infrastructure Unlike the Fog Computing scenario in Section 6.3, where a Fog network considers devices to switch their online or availability status (e.g., *Fog Node 2*), we now consider three leased VMs throughout the evaluation period. The corresponding Cloud resource infrastructure with the fixed amount of VMs is depicted in Figure 6.9. Herein, we consider the *VISP Runtime Cloud* as single Runtime that manages the VMs with the flavours *m1.medium*, *m2.medium*, and *m1.xlarge*. The cost of the VMs increase

with the amount of provided capacities, whereas *m1.medium* cost 10.5 CU/s. The more powerful instances *m2.medium* and *m1.xlarge* cost 15.5 CU/s and 20.5 CU/s respectively. Structural changes of the test bed are not considered throughout the 50 minutes evaluation period. This enables to focus on the placement changes that are caused by changing monitoring data as e.g., network delay, availability of resources, and processing durations of the operators.

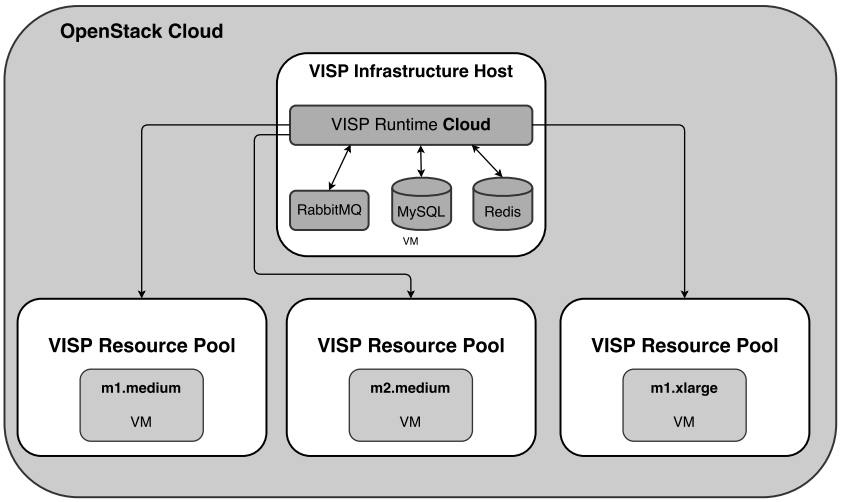


Figure 6.9: Cloud Resource Infrastructure

Topology Specification Listing A.2 specifies the topology that is used in this scenario. The general topology structure remains unchanged but the allowed locations of operators are adapted to the *VISP Runtime Cloud*. For this, we consider *128.130.172.182/** as value of the *allowedLocations* attribute for all operators to specify that there are no placement restrictions in the given resource infrastructure. The *concreteLocation* attribute is chosen to distribute the operators equally at the start of the evaluation period.

6.4.2 Full Model

In this scenario we consider the *Full Model* to optimize all involved QoS attributes with equal importance.

Evaluation Hypothesis Similarly to the Fog Computing scenario, we analyze the performance of QoS attributes. For this, we distinguish between a response time- and availability-oriented hypothesis, as well as a cost-oriented hypothesis:

> *H4: In an environment with fixed computing resources, the dynamic ODR approach is better than the static approach with respect to (a) response time and (b) availability.*

H5: In an environment with fixed computing resources, the sum of all cost of the dynamic ODR approach is less than the cost of the static approach.

These hypotheses are adapted from *H1* and *H2*, whereas *H4* and *H5* neglect Fog characteristics and only consider fixed computational resources.

Evaluation Results To start with the response time evaluation, we consider Figure 6.10 with a solid and a dashed line for the respective dynamic and static ODR approaches. The dynamic approach starts with a response time that is up to 13% faster than the one of the static approach (5.5% on average). With fluctuations still below the static line, the dynamic line finally approaches and marginally surpasses the baseline. This behavior can be explained by referring to the equal weights which are used in the objective function (see Equation 4.10 and Equation 4.23). Hence, the dynamic approach achieved to maximize the objective function by focusing on response time improvements at the beginning. In contrast, the static approach initially achieved the highest total score by focusing on the cost terms of the objective function, as we will discuss in this section.

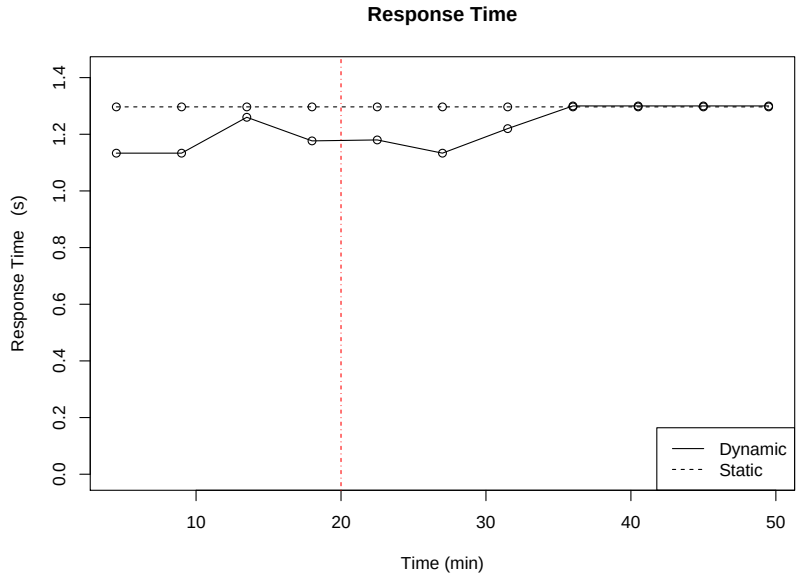


Figure 6.10: Response Time in the Full Model Cloud Scenario

The availability evaluation is presented in Figure 6.11. Here, we can observe that the dynamic line faces a very fluctuating history in the observed 50 minutes. In contrast, the static approach does not face any availability changes over the run time and remains stable at the 0.59 mark. In only 3 out of 11 data points the dynamic approach is above the baseline. Especially, after 20 minutes no placement can be computed that provides

an availability above the static line. This can be due to placement decisions that are made to satisfy any other used QoS criteria.

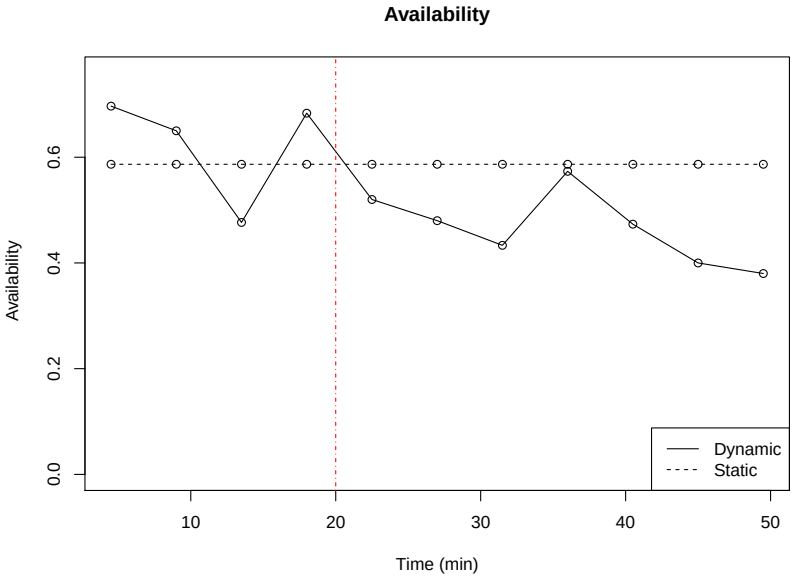


Figure 6.11: Availability in the Full Model Cloud Scenario

In Figure 6.12 the cost of the compared approaches is depicted. The static approach optimizes placements with enactment cost constantly at a level of 12.8 CU/s after the optimization cost is set to 0 when the placements have been optimized initially. In contrast, the dynamic total cost changes between 18.5 CU/s and 21 CU/s. It should be noted that the dynamic total cost appears to differ only by the optimization cost. This is due to very small migration cost. However, the additional relative cost for optimizing placements throughout the run time amounts to 6.47 CU/s, i.e., 49%, as shown in Table 6.6. For this, we can see that only the response time is improved, which leads to deficiencies in the remaining QoS attributes.

Table 6.6: Result Metrics for the Full Model Cloud Scenario

	Static	Dynamic
Average Response Time (sec)	1.29 ($\sigma = 0.18$)	1.22 ($\sigma = 0.1$)
Average Availability	0.58 ($\sigma = 0.16$)	0.52 ($\sigma = 0.03$)
Enactment Cost (CU/s)	12.82 ($\sigma = 2.23$)	16.54 ($\sigma = 1.22$)
Migration Cost (CU/s)	-	7.03 ($\sigma = 3.15$)
Optimization Cost (CU/s)	0.27 ($\sigma = 0$)	3 ($\sigma = 0$)
Total Cost (CU/s)	13.09 ($\sigma = 2.23$)	19.57 ($\sigma = 1.21$)
Cost Savings (CU/s)	-	-6.47 ($\sigma = 2.91$)
Average Total Score	0.85 ($\sigma = 0.03$)	0.77 ($\sigma = 0.06$)

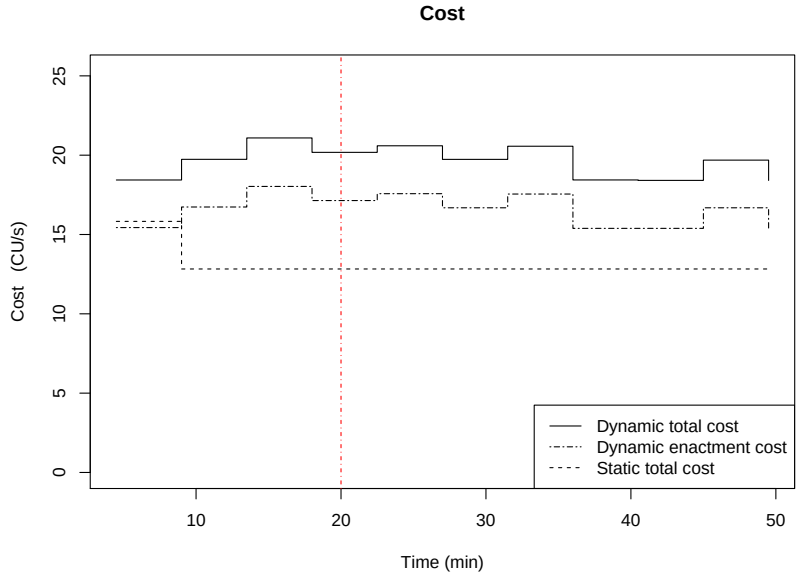


Figure 6.12: Cost in the Full Model Cloud Scenario

To verify the hypothesis of this scenario, we start with *H4*. Considering the response time results, we can observe that the dynamic ODR approach performs better at the beginning and nearly equally at the end of the evaluation period. This satisfies condition (a) of the *H4*. Nevertheless, the availability results have been partially better but most data points have shown that the static approach does better with respect to that specific QoS criteria. For this, we consider the improvements of the response time as the trade-off that has been made at the cost of availability. Therefore, condition (b) is not satisfied. Overall, this leads to a rejection of the given hypothesis *H4*. For hypothesis *H5*, we have compared the total cost. Herein, it can clearly be observed that there is no cost advantage of the dynamic approach. Additional cost of 49% has to be considered. Therefore, we also reject *H5* for the current scenario of fixed VMs in the Cloud. This implies that for a beneficial dynamic optimization, it needs a changing environment. In this case, it is possible that the improvements in respective QoS attributes outweigh the additional optimization cost.

6.5 Comparison of Fog and Cloud Scenarios

To analyze the difference between the Fog and the Cloud scenarios, we compare the respective *Full Model* scenarios as presented in Section 6.3.2 and Section 6.4. In the Fog scenario we have observed significant improvements in response time that amount to 18% (260 ms) on average. In contrast, the response time in the Cloud has been improved by 5.5% (70 ms) on average. For this, we conclude that the detection of structural

changes, introduced with *Fog Node 2* in the Fog scenario, makes a significant difference. This is because any change in the network that is detected, creates an opportunity for improvements to be exploited by the dynamic ODR Reasoner. So, the Cloud scenario provides an environment that is too stable to achieve major improvements. This is also a reason for the poor performance of ODR in the Cloud scenario with respect to the cost criteria. Herein, the additional optimization cost (3 CU/s) can not be compensated by finding cheaper placements and leads to an increase in total cost by 6.47 CU/s, i.e., 49%, which is invested in response time improvements. Nevertheless, we assume that there is still potential for dynamic optimization approaches in Cloud scenarios, when the resource cost changes over time. This creates opportunities that need to be dynamically re-evaluated as performed in the ODR Reasoner.

6.6 Discussion of Data Ingestion Pattern

In Section 6.2.1 we refer to data tuples that are ingested into the DSP topology. For this we consider a varying load in the form of a sinus function. Interestingly, we have not observed any response time pattern that is similar to this. To explain this behaviour we refer to the definition of the response time in the optimization model (see Equations 4.1-4.4). We now analyze the parameters that influence the response time and its components, i.e., operator response time (see Equation 4.3) and network delay (see Equation 4.4).

The first parameter is the execution time per data tuple ET_i as used in the operator response time component. It is set to a fixed value at the beginning of the evaluation period as described in Section 6.2.2. Since ET_i does not depend on current placements, it is not updated by the ODR after the *Warm-up Phase* (see Section 4.4.5). Hence, ET_i does not change the response time during the evaluation progress and therefore does not capture the sinusoidal data load.

The second parameter is the delay $d_{(u,v)}$ that is considered in Equation 4.4. In the evaluation, $d_{(u,v)}$ is set to the measured network delay between the corresponding resource nodes u, v. For this, the network delay can be seen as an estimator for $d_{(u,v)}$ without considering the time when tuples are queued before processing (see Section 5.2). Therefore, $d_{(u,v)}$ does not capture the sinusoidal data load.

The last parameter to be considered refers to S_u in Equation 4.3. For this, we can see that if operator i is migrated to resource node u with a different speedup S_u, then the operator response time changes. A replacement like this, can be triggered by e.g., the constraint that limits the tuple processing duration $T_{(actual,i)}$ as presented in Equation 4.26. If $T_{(actual,i)}$ on the left-hand side is greater than the maximum expected processing duration $T_{(max,i)}$ on the right-hand side, then the speedup S_u in the denominator has to be increased. The consequence is a replacement that is based on a varying load of ingested tuples reflected by $T_{(actual,i)}$.

We arrive now at a point where we have found a parameter, i.e., S_u, that considers the varying tuple load in the system. So, it could influence the response time throughout the

evaluation period. Nevertheless, this is not the case due to the following two reasons. First, S_u is limited to three discrete values, i.e., 1 (sp_{small}), 2 (sp_{medium}), and 4 (sp_{large}). Therefore, corresponding changes in the response time can be rather coarse-grained, which do not reflect a sinus pattern. Second, the constraint in Equation 4.26 considers parameters, that have not forced the ODR Reasoner adequately to change the response time to a sinusoidal form. We conclude that the pattern of the data ingestion load needs to be considered in a dedicated QoS criteria in order to be reflected (e.g., network load as used in [10]). Alternatively, the response time can be remodelled with parameters that reflect the changing data load more precisely.

6.7 Summary

In this chapter we presented the evaluation setup as well as the execution of a chosen DSP topology in different scenarios. The Fog computing scenarios included a varying amount of resources with different latencies in the network. Starting with the *Full Model*, promising results with an improvement of 31.5% for the response times have been observed. This was achieved by optimizing operator placements dynamically instead of computing static placements at the beginning of a 50 minutes period. For availability concerns, improvements with respect to the baseline were observable but not enough to state that the dynamic approach is better in all cases. However, cost savings of 8.8% have been observed when dynamic replacements were considered. The special case with the *Response Time Model*, clearly has shown advantages of pure response time optimization. At the end of the evaluation period, the dynamic ODR Reasoner approach decreased the response time by 33.5%. Considering the Cloud scenario, we have seen aspects (i.e., response time) which could have been improved by the dynamic ODR Reasoner, since it yielded better results than the corresponding static approach. Nevertheless, the stable environment of fixed resources (i.e., three VMs) does not have the potential to improve all QoS attributes as this is the case for a changing network structure or at least a network with varying conditions (e.g., latencies). In the next chapter, we further draw conclusions considering the evaluation results and hypothesis, besides answering the identified research questions.

CHAPTER 7

Discussion & Conclusion

The optimization of operator placements is an important method to improve the QoS of DSP topologies. In the literature research we identified various fields for processing on-line data streams to gain valuable insights (see Section 2.3). For this, we have focused on the IoT and provided a motivational scenario of the manufacturing domain in Chapter 1. The proposed ODR approach is based on an ILP optimization approach from Cardellini et al. [10] which has been identified as the most appropriate one. Considering this, ODR is a centralized ILP optimization approach that is dynamically applied. It aims for finding a global optimum with respect to multiple QoS criteria. The main difference to existing approaches is the combination of ILP optimization for the Fog environment and the dynamic reconfiguration of operator placements. ODR reacts to structural changes as well as to load changes in the Fog network. To investigate the impact of operator placement for the QoS of DSP topologies, we have evaluated our proposed ODR approach in a Cloud and simulated Fog Computing environment. We have demonstrated that our approach exploits these changes to provide QoS improvements. Although the degree of these improvements varies and strongly depends on the resource infrastructure, we think that ODR can be used beneficially in heterogeneous networks. The following chapter answers the identified research questions from Section 1.4 and discusses the limitations of our approach. To conclude this work, we outline future research possibilities with respect to DSP operator placement problems.

7.1 Discussion of Research Questions

RQ1 Which criteria are relevant for an operator (re)placement problem?

In Chapter 4 we have identified different QoS attributes that are incorporated into the ODR optimization model based on the literature review of Chapter 3. For this, we consider operator and network latency, enactment cost, resource availability, and operator replacement cost (i.e., migration cost).

The most prominent criterion used in literature is network latency. The network latency is generally considered to be minimized in centralized as well as decentralized algorithms. In addition to the network latency, Cardellini et al. [10, 15] also incorporated operator latency to the respective system models. To consider the overall latency of the DSP topology, the operator and network latencies are added up along all DSP paths. The maximum latency of all DSP topology paths is then assigned to the overall latency.

The resource availability is used in some approaches, that consider Fog Computing infrastructures. For these heterogeneous resources, operators could be located in highly diverse environments with different software and hardware conditions. To account for this, the availabilities of hosts as well as the availabilities of network links are vital. The overall availability of a DSP topology is then considered as the product of these resource availabilities [10].

The enactment cost is a criterion that is relevant in environments with differently priced resources. For this, Cloud Resources are typically considered. As indicated by Vaquero et al. [4], it can be assumed that business models for Fog resources will cover monetisation aspects of Fog resources. We conclude that similar to the Cloud, Fog Resources could be leased accordingly. To decrease the overall cost of DSP topologies, the cost for enacting their operators constitutes the major part and have to be minimized.

Another cost component is the operator placement cost. For this, migration cost has to be taken into account when it comes to replacements as it is the case in our dynamic ODR approach. Although we consider the operator replacement cost from a monetary point of view, other methods (e.g., threshold-based workload limitation) are used also in literature for minimizing the deployment changes as discussed in [74].

To go beyond the criteria that have been considered in this work, we refer to the network load. The network load is based on network delay and used bandwidth. Therefore, this criterion can be seen as an extension to the latency criterion.

RQ2 How can an operator placement problem be defined?

In Section 3.3.2 we discussed different optimization possibilities with respect to DSP. As it has been presented, placement optimization is a prominent method to improve the performance of a DSP topology. For this, Hirzel et al. [60] defined the operator placement problem as the problem of assigning DSP operators to hosts in a network. Placement restrictions and identified QoS criteria can complicate the optimization to be a time consuming and resource intensive task.

In Chapter 4 we provided design for the ODR optimization approach. Based on our system model in Section 4.2, we have introduced a formulation of the operator placement problem by considering an objective function and several constraints (see Section 4.3.2). We incorporated the identified QoS criteria latency (i.e., response time), availability, enactment cost, and migration cost into the objective function. After normalizing them, they are weighted and added up according to the SAW [73] approach. Defining the objective function in that way, enables to solve the operator placement problem with respect to multiple heterogeneous criteria. The weights represent the importance of the respective criteria. The second part of operator placement problem definition includes constraints that limit the operators to be placed on hosts with enough resources capacities. Furthermore, other constraints ensure the correct formal behavior of the placement process, e.g., an operator can be deployed to only one host instead of splitting it up.

We have further added another aspect for defining operator placement problems, which is the dynamic consideration of changes. Since dynamic environments such as Fog Networks change continuously regarding their structures and network conditions, operator placement problems are not limited to a fixed point in time. Moreover, operator replacements have to be considered throughout the enactment of DSP topologies. Therefore, changing situations are an integral part of operator placement problems. In literature, this aspect is known, but not always incorporated into respective optimization approaches. Nevertheless, we found that especially decentralized algorithms consider dynamic replacements and therefore take continuous updates of the environment into account.

RQ3 How can our optimization approach be realized as software and be integrated into established systems?

In Chapter 4 and Chapter 5 we have focused on the design and implementation of ODR respectively. To answers research question RQ3, we discuss the main results of these chapters. We describe how the ODR optimization approach is realized and integrated into the VISP ecosystem.

Based on the created ILP optimization model (see Section 4.3.2), we have designed the ODR Reasoner software (see Section 4.4). It consists of an *API* to receive optimization tasks from VISP Runtimes and a *VISP Client* to retrieve metric updates of the resource infrastructure, i.e., Cloud or Fog network. The obtained metrics are used to derive the parameters that are required in the optimization model. After a request is received, a *Scheduler* component periodically invokes the CPLEX-based *ILP Model Solver* that solves the placement problem. The resulting operator placements are propagated to the VISP Runtimes for executing the changes. Besides an *ILP Model Solver*, other key components are the *Metric Provider* and the *Resource Manager* to facilitate the periodic updates of parameters. The former one parses operator and network condition metrics, while the latter one detects resource changes like e.g., a Fog node that switched its availability status.

The ODR implementation is a Java-based service that is loosely coupled to VISP Runtimes. The communication protocol is HTTP, which is supported by software frameworks, e.g., Spring Boot. Data exchange between the involved systems is applied in two ways. Data that is received from VISP is transported in JSON objects as defined in the description of involved VISP runtime endpoints (see Section 5.3). Conversely, the data sent to VISP is transported as a VISP Topology Description Language file containing the placement updates and possible scaling instructions.

RQ4 How does the optimization compare against baseline approaches?

In Chapter 6 we have evaluated the (dynamic) ODR approach to investigate if improvements can be achieved compared to the baseline. For this, we considered the static optimization approach. In contrast to the dynamic ODR Reasoner, the baseline solves the operator placement problem only at system startup. Three scenarios have been evaluated, whereas each is based on the topology defined in Section 6.1.2). In two scenarios the resource infrastructure is a Fog-like environment, while in the third scenario a Cloud environment is considered. To provide a benchmark between ODR and the baseline, we summarized the outcome of all three scenarios in Table 7.1. The columns *Response Time*, *Availability* and *Cost* compare the two approaches by showing if the ODR approach is better throughout the evaluation period. This is indicated with '✓', while '-' indicates that there are situations in which the static approach performs better. This corresponds to the hypotheses that have been used during evaluation, whereas '✓' stands for the acceptance of the respective hypothesis or at least for satisfying the respective condition, i.e., (a) and (b) in *H1* and *H4* (see Section 6.3). The columns *Response Time Decrease* and *Cost Savings* provide information about the relative improvement in the respective QoS criterion.

In the first scenario, i.e., Full Model Fog Scenario (see Section 6.3) we found that the response time of ODR has significantly decreased by up to 29.5% when new Fog resources are added. Hence, ODR exploits the option to change the placement structure for achieving a better QoS. In contrast, due to the inflexibility, the baseline approach misses this chance. Considering the availability, we have observed a good performance of the ODR approach in the majority of measured time points. The cost savings of 8.8% have been achieved by the ODR approach over the whole evaluation period. In this scenario, we have demonstrated that the overall result is satisfying, although the availability condition (b) within hypothesis *H1* could not have been accepted. So, we still see potential for improvements in this field.

The second, more specialized scenario considers solely the optimization of the response time. For this, the ODR approach has decreased the response time by up to 31.5% compared to the static approach. Herein, we conclude that neglecting other QoS criteria can lead to even better results for QoS criteria that have received higher weights. Hence, it can be considered as a feature of the ODR Reasoner to define the importance of provided QoS criteria by adapting their weights.

The third scenario refers to a Cloud environment with used resources that are not varying, since the amount of Resource Pools and assigned VMs is fixed. The response time has been decreased clearly by the dynamic ODR approach, although the maximum of 13% is much lower than the corresponding results in the Fog scenarios. Nevertheless, this result is caused only by changing conditions in the resources and not by adding or removing resources. In contrast, additional cost of 49% and poor availability have to be considered for decreasing the response time in exchange. For this, we conclude, that not all QoS criteria could have been improved in the Cloud scenario. Therefore, we found that in rather stable environments, improvements can be made with respect to selected QoS criteria, but dynamic optimization can come with deficiencies for other QoS criteria and cost in particular.

Table 7.1: Comparison of the Dynamic ODR with the Static Baseline

	Response Time	Response Time Decrease	Availability	Cost	Cost Savings
Full Model Fog	✓	31.5%	-	✓	8.8%
Response Time Model Fog	✓	33.5%	n.a.	n.a.	n.a.
Full Model Cloud	✓	13%	-	-	-49%

7.2 Limitations

The presented ODR approach comes with limitations that can be identified in different areas. First, the placement problem is NP-hard. Therefore, an increasing problem size, i.e., larger Fog Networks and complex topologies, can make it difficult to solve it in reasonable time. For this, it has to be considered that ODR is a centralized approach, which solves the problem by incorporating all parameters from VISP Runtimes. Besides the resource nodes V_{res}, we see the implemented full mesh representation of the resource graph G_{res} as a complexity driver that increases the number of edges E_{res}. So if, multiple decentralized VISP Runtimes enact a DSP topology on a large number of Resource Pools and forward their metrics continuously to the ODR reasoner, it may not find a solution. This situation is even more complicated in periodic optimization, since optimization tasks may not finish in the given period, which can lead to results based on outdated parameter and a delay in the next period. Furthermore, the resources of the ODR Reasoner might not suffice to serve multiple optimization tasks in parallel.

Another limitation is the completeness, accuracy, and update cycle of the metrics that are sent to the ODR Reasoner. So, if the metrics are not updated on a regular basis, or are rather coarse grained, the used parameters have to be estimated. If this estimation does not reflect the current situation in the resource infrastructure, the operator placements that are suggested to the VISP Runtimes might not be optimal.

The weights of the objective function (see Equations 4.10 and 4.23) directly influence the quality of the results. So, the issuer of an optimization task has to perform a fine tuning in testing runs before the best weights configuration is found and the actual optimization can be started on a live system.

7.3 Future Work

This work has focussed on DSP operator placements with dynamic reconfiguration. The aspect of dynamicity in connection with the aspect of finding a global optimum with ILP models contains many challenges. Furthermore, the consideration of Fog Networks for hosting DSP operators can be very promising to e.g., reduce latency and save cost.

In this work we made use of a persistence strategy according to Woodside et al. [74]. We have considered migration cost to limit the amount of operator replacements in each turn. This can help in large Fog networks to keep the reorganization effort to a minimum. If further persistence strategies can be integrated, it could decrease the complexity of the optimization problem as well. Hence, a dynamic ILP approach that aims for achieving a global optimum might then be able to perform well for large Fog networks. Beside persistence strategies, other heuristics, which reduce the solution time for finding a global optimum, need to be evaluated.

In literature we found only purely centralized algorithms that use ILP models for operator placement optimization (see Section 3.3.10). For this, it can be beneficial to consider a composition of multiple ILP models that are solved in a decentralized way for an assigned region in the Fog network. A centralized optimizer could then consider the results from each region and make final adaptations that are rolled out to the entire infrastructure. This hybrid approach can make a compromise to reduce the time of solving the optimization model and at the same time it can achieve optimal results that can be very close to a potential global optimum.

To facilitate this hybrid optimization, we could use the resources provided in the Fog. So, we do not only consider a fixed decentralized network of solving units that optimize the placement problem for their assigned region. Moreover, it can be beneficial to exploit also the resources of temporarily available Fog nodes. For this, a future research possibility is to investigate how Fog resource provisioning frameworks can be used for distributing and coordinating optimization tasks.

As discussed in the literature review (see Section 3.3.6), the network load is a QoS criterion that can be incorporated into the optimization of operator placements. An extended version of the ODP approach [10] has considered this criterion in an ILP model, but just focussed on static optimization. For this, new possibilities and improvement potentials have to be evaluated with respect to dynamic consideration of the network load in an ILP model.

APPENDIX A

Evaluation Topologies

A.1 Fog Topology

Listing A.1: Fog Topology

```
1  $source = Source() {
2   concreteLocation = 128.130.172.183/fogPool1,
3   type             = source,
4   outputFormat     = "temperature data from machine sensor",
5   expectedDuration = 15,
6   size = small
7  }
8
9  $step1 = Operator($source) {
10  allowedLocations = 128.130.172.183/*,
11  concreteLocation = 128.130.172.183/fogPool1,
12  inputFormat      = step1,
13  type             = step1,
14  outputFormat     = step2,
15  size             = small
16 }
17
18 $step2 = Operator($step1) {
19  allowedLocations
20        = 128.130.172.182/* 128.130.172.183/* 128.130.172.217/*,
21  concreteLocation = 128.130.172.182/cloudPool2,
22  inputFormat      = step1,
23  type             = "step2",
24  outputFormat     = "step3",
25  size             = small,
26  expectedDuration = 15
27 }
28
29 $step3 = Operator($step1) {
30  allowedLocations
```

```
31          = 128.130.172.182/* 128.130.172.183/* 128.130.172.217/*,
32   concreteLocation = 128.130.172.182/cloudPool1,
33   inputFormat      = step1,
34   type             = "step3",
35   outputFormat     = "step3",
36   size             = small,
37   expectedDuration = 15
38  }
39
40  $step4 = Operator($step1) {
41   allowedLocations
42          = 128.130.172.182/* 128.130.172.183/* 128.130.172.217/*,
43   concreteLocation = 128.130.172.182/cloudPool2,
44   inputFormat      = step1,
45   type             = "step4",
46   outputFormat     = "step3",
47   size             = small,
48   expectedDuration = 15
49  }
50
51  $log = Sink($step2, $step3, $step4) {
52   concreteLocation = 128.130.172.182/cloudPool2,
53   inputFormat      = "transformed data",
54   type             = "log-type operator"
55  }
```

A.2 Cloud Topology

Listing A.2: Cloud Topology

```
 1  $source = Source() {
 2   concreteLocation = 128.130.172.182/cloudPool0,
 3   type             = source,
 4   outputFormat     = "temperature data from machine sensor",
 5   expectedDuration = 15,
 6   size = small
 7  }
 8
 9  $step1 = Operator($source) {
10   allowedLocations = 128.130.172.182/*,
11   concreteLocation = 128.130.172.182/cloudPool0,
12   inputFormat      = step1,
13   type             = step1,
14   outputFormat     = step2,
15   size             = small
16  }
17
18  $step2 = Operator($step1) {
19   allowedLocations = 128.130.172.182/*,
20   concreteLocation = 128.130.172.182/cloudPool1,
21   inputFormat      = step1,
22   type             = "step2",
```

```
23   outputFormat      = "step3",
24   size              = small,
25   expectedDuration = 15
26  }
27
28  $step3 = Operator($step1) {
29   allowedLocations = 128.130.172.182/*,
30   concreteLocation = 128.130.172.182/cloudPool1,
31   inputFormat       = step1,
32   type              = "step3",
33   outputFormat      = "step3",
34   size              = small,
35   expectedDuration = 15
36  }
37
38  $step4 = Operator($step1) {
39   allowedLocations = 128.130.172.182/*,
40   concreteLocation = 128.130.172.182/cloudPool2,
41   inputFormat       = step1,
42   type              = "step4",
43   outputFormat      = "step3",
44   size              = small,
45   expectedDuration = 15
46  }
47
48  $log = Sink($step2, $step3, $step4) {
49   concreteLocation = 128.130.172.182/cloudPool0,
50   inputFormat       = "transformed data",
51   type              = "log-type operator"
52  }
```

APPENDIX B

Identified Requirements and Parametrization Phases

Table B.1: Features and Corresponding Technical Tasks. Part 1.

Id	Feature or Work Item	Description	Task
1	SW Design Setup	Transformation of software design to a first code basis	• Setup Web Application Framework • Create packages, interfaces and stubs for key classes
2	Entity Model	Creation of entity classes	• DSP graph • Resource graph • Other parameter and DTOs
3	API	RESTful Interface for the ODR Reasoner	• parse request for adding new optimization tasks • parse initial DSP topology, resource infrastructure and all optimization relevant parameter to store it in the data context. Then trigger parametrization. • receive the command for starting or aborting an optimization task

Table B.2: Features and Corresponding Technical Tasks. Part 2.

Id	Feature or Work Item	Description	Task
4	Metric Provider	Creation of Metric Providers for receiving monitoring data	• Simulation Metric Provider (parameters assumed) • VISP Metric Provider – initial parameter estimation – parse received metric updates – recompute parameter based on new metrics and existing parametrized model
5	Resource Manager	Creation of a resource manager that is in charge of initializing and updating the DSP and resource model	• parse received resource change updates and adapt the resource graph model • recompute parameters for the adapted model
6	Scheduling	Periodical optimization and reconfiguration scheduling	• retrieve current parametrized model from Data Context • invoke the ILP Solver • invoke the suitable persistence strategy [74] • invoke reconfiguration

Table B.3: Features and Corresponding Technical Tasks. Part 3.

Id	Feature or Work Item	Description	Task
7	Persistence Strategy	Limiting the degree of replacement actions to reduce complexity for the optimization problem	• use heuristics to reduce the part of the resource and/or DSP graph that has to be adapted
8	ILP Solving	Solving the model with *IBM CPLEX Optimizer*[1]	• transform the designed optimization model to JAVA (compile) • solve the compiled model and extract the solution [74]
9	Reconfiguration	Perform the reconfiguration of the placement dynamically (replacement)	• Consider the new placement and invoke the VISP Runtime Callback to upload the changed VISP Topology Description Language File.
10	Reporting	Reporting of new placements and current QoS attributes	• Periodic refreshing of reported values • Show placement (assignment of DSP operators on a graph of resources) • Show current QoS attribute values • Export QoS attribute values as spreadsheet
11	Simulation	Simulate the VISP Runtime as component that uses the ODR Reasoner for starting optimization tasks and receiving replacement instructions	• Define simulation scenarios • Invoke the ODR Reasoner API to start the optimization with DSP and resource graphs as defined in the scenario • Send metric updates periodically as defined in the simulation scenario • Send resource changes at defined time points
12	Integration	Integrate VISP with the ODR Reasoner	• Identify and integrate exchanged DTOs • Call HTTP interface of VISP for retrieving data • Parse VISP specific input data (e.g., VISP Topolgy Description File) • Prepare topology file upload for updating placement decisions

Table B.4: Determination of Parameters in Different Phases

Phase	Parameters	Description
Creation Phase	DSP toplogy graph (G_{dsp})	• Nodes (V_{dsp}) – required CPU frequency per core ($P_{(CPU,i)}$) – required cores ($P_{(Cores,i)}$) – required memory ($P_{(Mem,i)}$) – required storage ($P_{(HD,i)}$) – image size (s_i) • Edges (E_{dsp})
	Resource graph (G_{res})	• Nodes (V_{res}) – availability (A_u) – enactment cost per second (C_u) – available CPU power per core ($P_{(CPU,u)}$) – available cores ($P_{(Cores,u)}$) – available memory ($P_{(Mem,u)}$) – available storage ($P_{(HD,u)}$) – speedup of host compared to the reference processor (S_u). This parameter is derived using defined resource classes *small*, *medium*, *large*. – data rate to the VISP Marketplace ($b_{(M,u)}$) • Edges (E_{res}) – data rate (bandwidth) ($b_{u,v}$) – availability ($A_{u,v}$) – delay ($d_{(u,v)}$)
	QoS attribute weights	weights for importance of QoS attributes in the objective function with respect to availability w_a, latency w_r, enactment cost $w_{c_{op}}$, and migration cost $w_{c_{mig}}$
Warm-up Phase	Execution time per data tuple for each operator (ET_i)	Each operator is hosted on a system with reference processors, where ET_i is measured during a test run. Nevertheless, the user can skip this test run and assign default values too.
Update Phase	All parameters from the starting phase and the warm-up phase.	These parameters are pulled periodically from VISP Runtimes. If the returned parameters deviate from their previous or default values, an updated is performed. - Considering the delay $d_{(u,v)}$ between nodes $u, v \in V_{res}$, it could be the case that no information from VISP Runtimes can be retrieved for all $(u, v) \in E_{res}$. Therefore it needs to be approximated with $\frac{\sum_{(i,j)\in E_{res}} d_{(i,j)}}{\lvert E_{res}\rvert}$.

List of Figures

List of Tables

Listings

Acronyms

API Application Programming Interface. 15

BPMS Business Process Management System. 37

CDN Content Delivery Network. 17

CEP Complex Event Processing. 21

CoAP Constrained Application Protocol. 13

CU Currency Unit. 79

DBMS Database Management System. 20

DDS Distributed Data Store. 39

DHT Distributed Hash Table. 35

DSP Data Stream Processing. xi, 1, 9, 20

DTO Data Transfer Object. 65

EEG Electroencephalography. 17

HMI Human-to-Machine Interaction. 22

IaaS Infrastructure as a Service. 15

ILP Integer Linear Program. 5, 31

IoT Internet of Things. xi, 1, 9

IP Internet Protocol. 9

JSON JavaScript Object Notation. 68

JVM Java Virtual Machine. 15

KPI Key Performance Indicator. 2, 38

M2M Machine-to-Machine Interaction. 22

MQTT Message Queue Telemetry Transport. 13

NIC US National Intelligence Council. 10

NIST National Institute of Standards and Technology. 15

ODP Optimal DSP Placement. 5

ODR Optimal DSP Replacement. 5, 41

OS Operating System. 29

OSI OSI Model. 12

PaaS Platform as a Service. 15

PE Processing Element. 26

PEC Processing Element Container. 26

QoS Quality of Service. 2

REST Representational State Transfer. 56

RFID Radio-Frequency Identification. 9

RQ Research Question. 5

SaaS Software as a Service. 15

SAW Simple Additive Weighting. 48, 51

SLA Service Level Agreement. 38

SOA Service Oriented Architecture. 11

SPADE Stream Processing Application Declarative Engine. 26

SSH Secure Shell. 78

VISP VIenna ecosystem for elastic Stream Processing. 4

VM Virtual Machine. 38

WSN Wireless Sensor Network. 12

Bibliography

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Generation Computer Systems (FGCS)*, vol. 29, no. 7, pp. 1645–1660, 2013.

[2] H. Lasi, P. Fettke, H.-g. Kemper, T. Feld, and M. Hoffmann, "Industry 4.0," *Business and Information Systems Engineering*, vol. 6, pp. 239–242, 08 2014.

[3] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog Computing and its Role in the Internet of Things," in *Proceedings of the 1st edition of the MCC workshop on Mobile cloud computing*, pp. 13–16, ACM, 2012.

[4] L. M. Vaquero and L. Rodero-Merino, "Finding Your Way in the Fog: Towards a Comprehensive Definition of Fog Computing," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 5, pp. 27–32, 2014.

[5] C. Hochreiner, M. Vögler, S. Schulte, and S. Dustdar, "Stream Processing for the Internet of Things," in *9th International Conference on Cloud Computing (CLOUD)*, pp. 100–107, IEEE, 2016.

[6] B. Gedik, H. Andrade, K.-L. Wu, P. S. Yu, and M. Doo, "SPADE: The System S Declarative Stream Processing Engine," in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pp. 1123–1134, ACM, 2008.

[7] A. V. Dastjerdi and R. Buyya, "Fog Computing: Helping the Internet of Things Realize Its Potential," *Computer*, vol. 49, no. 8, pp. 112–116, 2016.

[8] R. Buyya and A. V. Dastjerdi, *Internet of Things: Principles and Paradigms*. Elsevier, 2016.

[9] S. Yi, C. Li, and Q. Li, "A Survey of Fog Computing: Concepts, Applications and Issues," in *Proceedings of the 2015 Workshop on Mobile Big Data*, Mobidata '15, pp. 37–42, ACM, 2015.

[10] V. Cardellini, V. Grassi, F. Lo Presti, and M. Nardelli, "Optimal operator placement for distributed stream processing applications," in *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems (DEBS)*, pp. 69–80, ACM, 2016.

[11] C. Hochreiner, M. Vögler, P. Waibel, and S. Dustdar, "VISP: An Ecosystem for Elastic Data Stream Processing for the Internet of Things," in *20th International Enterprise Distributed Object Computing Conference (EDOC)*, pp. 19–29, IEEE, 2016.

[12] V. Cardellini, V. Grassi, F. Lo Presti, and M. Nardelli, "On QoS-aware Scheduling of Data Stream Applications over Fog Computing Infrastructures," in *5th International Workshop on Management of Cloud and Smart City Systems 2015 (MOCS 2015)*, pp. 271–276, IEEE, 2015.

[13] V. Cardellini, V. Grassi, F. Lo Presti, and M. Nardelli, "Distributed QoS-aware scheduling in storm," in *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems (DEBS)*, pp. 344–347, ACM, 2015.

[14] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. Seltzer, "Network-aware Operator Placement for Stream-Processing Systems," in *22nd International Conference on Data Engineering (ICDE'06)*, pp. 49–49, IEEE, 2006.

[15] V. Cardellini, V. Grassi, F. L. Presti, and M. Nardelli, "Optimal Operator Replication and Placement for Distributed Stream Processing Systems," *SIGMETRICS Performance Evaluation Review*, vol. 44, no. 4, pp. 11–22, 2017.

[16] V. Cardellini, M. Nardelli, and D. Luzi, "Elastic Stateful Stream Processing in Storm," in *2016 International Conference on High Performance Computing Simulation (HPCS)*, pp. 583–590, 2016.

[17] S. Rizou, F. Durr, and K. Rothermel, "Solving the multi-operator placement problem in large-scale operator networks," in *Computer Communications and Networks (ICCCN), 2010 Proceedings of 19th International Conference on*, pp. 1–6, 2010.

[18] Y. Ahmad and U. Çetintemel, "Network-aware Query Processing for Stream-based Applications," in *Proceedings of the 30th International Conference on Very Large Data Bases - Volume 30*, VLDB '04, pp. 456–467, VLDB Endowment, 2004.

[19] B. Kitchenham, O. P. Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic Literature Reviews in Software Engineering–a Systematic Literature Review," *Information and Software Technology*, vol. 51, no. 1, pp. 7–15, 2009.

[20] D. Giusto, A. Iera, G. Morabito, and L. Atzori, *The Internet of Things: 20th Tyrrhenian Workshop on Digital Communications*. Springer Science & Business Media, 2010.

[21] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787 – 2805, 2010.

[22] D. INFSO, "Networked Enterprise & RFID INFSO G. 2 Micro & Nanosystems, in co-operation with the Working Group RFID of the ETP EPOSS, Internet of Things

in 2020, Roadmap for the Future [R]," *Information Society and Media, Tech. Rep*, 2008.

[23] W. Shang, Y. Yu, R. Droms, and L. Zhang, "Challenges in IoT Networking via TCP/IP Architecture," tech. rep., NDN Project, Tech. Rep. NDN-0038, 2016.

[24] N. Gershenfeld, R. Krikorian, and D. Cohen, "Internet of Things," *Scientific American*, 2004.

[25] A. Margara, J. Urbani, F. van Harmelen, and H. Bal, "Streaming the Web: Reasoning over Dynamic Data," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 25, pp. 24 – 44, 2014.

[26] D. Anicic, P. Fodor, S. Rudolph, and N. Stojanovic, "EP-SPARQL: A Unified Language for Event Processing and Stream Reasoning," in *Proceedings of the 20th International Conference on World Wide Web*, (WWW '11), pp. 635–644, ACM, 2011.

[27] E. D. Valle, S. Ceri, F. v. Harmelen, and D. Fensel, "It's a Streaming World! Reasoning upon Rapidly Changing Information," *IEEE Intelligent Systems*, vol. 24, no. 6, pp. 83–89, 2009.

[28] N. N, "Disruptive Civil Technologies: Six Technologies with Potential Impacts on US Interests out to 2025," 2008.

[29] A. Ilic, T. Staake, and E. Fleisch, "Using Sensor Information to Reduce the Carbon Footprint of Perishable Goods," *IEEE Pervasive Computing*, vol. 8, no. 1, pp. 22–29, 2009.

[30] P. Spiess, S. Karnouskos, D. Guinard, D. Savio, O. Baecker, L. M. S. d. Souza, and V. Trifa, "SOA-Based Integration of the Internet of Things in Enterprise Services," in *2009 IEEE International Conference on Web Services*, pp. 968–975, IEEE, 2009.

[31] International Organization for Standardization, "Information Technology — Open Systems Interconnection (OSI) — Transport Service Definition." ISO/IEC 8072, 1996.

[32] S. Bandyopadhyay, M. Sengupta, S. Maiti, and S. Dutta, "Role of Middleware for Internet of Things: A Study," *International Journal of Computer Science and Engineering Survey*, vol. 2, no. 3, pp. 94–105, 2011.

[33] M. Eisenhauer, P. Rosengren, and P. Antolin, "A Development Platform for Integrating Wireless Devices and Sensors into Ambient Intelligence Systems," in *2009 6th IEEE Annual Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks Workshops*, pp. 1–3, 2009.

[34] P. Kumar, S. Ranganath, H. Weimin, and K. Sengupta, "Framework for Real-time Behavior Interpretation from Traffic Video," *IEEE Transactions on Intelligent Transportation Systems*, vol. 6, no. 1, pp. 43–53, 2005.

[35] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," *IEEE Communications Surveys Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[36] H. Kagermann, "Chancen von Industrie 4.0 nutzen," in *Handbuch Industrie 4.0 Bd. 4*, pp. 235–246, Springer, 2017.

[37] H. Kopetz, *Internet of Things*, pp. 307–323. Boston, MA: Springer US, 2011.

[38] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," 2011.

[39] Z. Hill and M. Humphrey, "CSAL: A Cloud Storage Abstraction Layer to Enable Portable Cloud Applications," in *2010 IEEE 2nd International Conference on Cloud Computing Technology and Science*, pp. 504–511, IEEE, 2010.

[40] G. Lawton, "Developing Software Online With Platform-as-a-Service Technology," *Computer*, vol. 41, pp. 13–15, June 2008.

[41] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A View of Cloud Computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[42] I. Stojmenovic and S. Wen, "The Fog Computing Paradigm: Scenarios and Security Issues," in *Federated Conference on Computer Science and Information Systems*, pp. 1–8, 2014.

[43] B. Ottenwälder, B. Koldehofe, K. Rothermel, and U. Ramachandran, "MigCEP: Operator Migration for Mobility Driven Distributed Complex Event Processing," in *Proceedings of the 7th ACM International Conference on Distributed Event-based Systems*, (DEBS), pp. 183–194, ACM, 2013.

[44] A. V. Dastjerdi, H. Gupta, R. N. Calheiros, S. K. Ghosh, and R. Buyya, "Fog Computing: Principles, Architectures, and Applications," *CoRR*, vol. abs/1601.02752, 2016.

[45] Y. Cao, S. Chen, P. Hou, and D. Brown, "FAST: A Fog Computing Assisted Distributed Analytics System to Monitor Fall for Stroke Mitigation," in *IEEE International Conference on Networking, Architecture and Storage (NAS)*, pp. 2–11, IEEE, 2015.

[46] J. K. Zao, T. T. Gan, C. K. You, S. J. R. Méndez, C. E. Chung, Y. T. Wang, T. Mullen, and T. P. Jung, "Augmented Brain Computer Interaction Based on Fog Computing and Linked Data," in *International Conference on Intelligent Environments*, pp. 374–377, 2014.

[47] J. Zhu, D. S. Chan, M. S. Prabhu, P. Natarajan, H. Hu, and F. Bonomi, "Improving Web Sites Performance Using Edge Servers in Fog Computing Architecture," in *IEEE 7th International Symposium on Service-Oriented System Engineering*, pp. 320–323, IEEE, 2013.

[48] W. Shi and S. Dustdar, "The Promise of Edge Computing," *Computer*, vol. 49, no. 5, pp. 78–81, 2016.

[49] V. Gazis, A. Leonardi, K. Mathioudakis, K. Sasloglou, P. Kikiras, and R. Sudhaakar, "Components of Fog Computing in an Industrial Internet of Things Context," in *12th Annual IEEE International Conference on Sensing, Communication, and Networking - Workshops (SECON Workshops)*, pp. 1–6, IEEE, 2015.

[50] K. Saharan and A. Kumar, "Fog in Comparison to Cloud: A Survey," *International Journal of Computer Applications*, vol. 122, no. 3, 2015.

[51] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network Function Virtualization: State-of-the-art and Research Challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 236–262, 2015.

[52] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[53] J. Gama and P. P. Rodrigues, *Data Stream Processing*, pp. 25–39. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007.

[54] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, and R. Varma, "Query Processing, Resource Management, and Approximation in a Data Stream Management System –," in *CIDR*, pp. 245–256, 2003.

[55] P. Misra, Y. L. Simmhan, and J. Warrior, "Towards a practical architecture for the next generation internet of things," *CoRR*, vol. abs/1502.00797, 2015.

[56] "What Is Big Data? - Gartner IT Glossary - Big Data." `http://www.gartner.com/it-glossary/big-data`. (Accessed on 04/19/2017).

[57] G. Cugola and A. Margara, "Processing Flows of Information: From Data Stream to Complex Event Processing," *ACM Computing Surveys*, vol. 44, pp. 15:1–15:62, June 2012.

[58] D. C. Luckham and B. Frasca, "Complex Event Processing in Distributed Systems," tech. rep., Stanford University, 1998.

[59] K. Goodhope, J. Koshy, J. Kreps, N. Narkhede, R. Park, J. Rao, and V. Y. Ye, "Building LinkedIn's Real-time Activity Data Pipeline.," *IEEE Data Engineering Bulletin*, vol. 35, no. 2, pp. 33–45, 2012.

[60] M. Hirzel, R. Soulé, S. Schneider, B. Gedik, and R. Grimm, "A Catalog of Stream Processing Optimizations," *ACM Computing Surveys*, vol. 46, pp. 46:1–46:34, Mar. 2014.

[61] O. Skarlat, S. Schulte, M. Borkowski, and P. Leitner, "Resource Provisioning for IoT Services in the Fog," in *IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*, pp. 32–39, IEEE, 2016.

[62] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, *Fog Computing: A Platform for Internet of Things and Analytics*, pp. 169–186. Cham: Springer International Publishing, 2014.

[63] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: A Decentralized Network Coordinate System," *SIGCOMM Computer Communication Revue*, vol. 34, no. 4, pp. 15–26, 2004.

[64] G. T. Lakshmanan, Y. Li, and R. Strom, "Placement Strategies for Internet-Scale Data Stream Systems," *IEEE Internet Computing*, vol. 12, no. 6, pp. 50–60, 2008.

[65] S. Schulte, P. Hoenisch, S. Venugopal, and S. Dustdar, "Introducing the Vienna Platform for Elastic Processes," in *International Conference on Service-Oriented Computing*, pp. 179–190, Springer, 2012.

[66] P. Hoenisch, D. Schuller, C. Hochreiner, S. Schulte, and S. Dustdar, "Elastic Process Optimization–The Service Instance Placement Problem," tech. rep., Tech. Rep. TUV-1841-2014-01, Distributed Systems Group, Vienna University of Technology (2014).

[67] P. Hoenisch, D. Schuller, S. Schulte, C. Hochreiner, and S. Dustdar, "Optimization of Complex Elastic Processes," *IEEE Transactions on Services Computing*, vol. 9, no. 5, pp. 700–713, 2016.

[68] P. Hoenisch, C. Hochreiner, D. Schuller, S. Schulte, J. Mendling, and S. Dustdar, "Cost-Efficient Scheduling of Elastic Processes in Hybrid Clouds," in *IEEE 8th International Conference on Cloud Computing (CLOUD)*, pp. 17–24, IEEE, 2015.

[69] P. Hoenisch, S. Schulte, and S. Dustdar, "Workflow scheduling and resource allocation for cloud-based execution of elastic processes," in *IEEE 6th International Conference on Service-Oriented Computing and Applications (SOCA)*, pp. 1–8, 2013.

[70] S. Euting, C. Janiesch, R. Fischer, S. Tai, and I. Weber, "Scalable Business Process Execution in the Cloud," in *2014 IEEE International Conference on Cloud Engineering*, pp. 175–184, 2014.

[71] T. Heinze, L. Aniello, L. Querzoni, and Z. Jerzak, "Cloud-based Data Stream Processing," in *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems (DEBS)*, pp. 238–245, ACM, 2014.

[72] R. Moen and C. Norman, "Evolution of the PDCA Cycle," 2006.

[73] K. P. Yoon and C.-L. Hwang, *Multiple Attribute Decision Making: An Introduction*, vol. 104. Sage publications, 1995.

[74] M. Woodside, Z. Li, J. Chinneck, and M. Litoiu, "Adaptive Cloud Deployment using Persistence Strategies and Application Awareness," *IEEE Transactions on Cloud Computing*, 2015.

[75] R. H. Peterson, *Accounting for Fixed Assets*. J. Wiley, 2002.

[76] T. H. Luan, L. Gao, Z. Li, Y. Xiang, and L. Sun, "Fog Computing: Focusing on Mobile Users at the Edge," *CoRR*, vol. abs/1502.01815, 2015.