University Library

A gateway to Melbourne's research publications

Minerva Access is the Institutional Repository of The University of Melbourne

Author/s:
Amarasinghe, Gayashan Niroshana

Title:
Distributed data stream processing and task placement on edge-cloud infrastructure

Date:
2021

Persistent Link:
https://hdl.handle.net/11343/274838

Terms and Conditions:
Terms and Conditions: Copyright in works deposited in Minerva Access is retained by the copyright owner. The work may not be altered without permission from the copyright owner. Readers may only download, print and save electronic copies of whole works for their own personal non-commercial use. Any use that exceeds these limits requires permission from the copyright owner. Attribution is essential when quoting or paraphrasing from these works.

Distributed Data Stream Processing and Task Placement on Edge-Cloud Infrastructure

by

Gayashan Niroshana Amarasinghe

ORCID: 0000-0001-8238-413X

A thesis submitted in total fulfillment for the
degree of Doctor of Philosophy

in the
School of Computing and Information Systems
Melbourne School of Engineering
THE UNIVERSITY OF MELBOURNE

8th January 2021

Copyright © 2021 Gayashan Niroshana Amarasinghe

All rights reserved. No part of the publication may be reproduced in any form by print, photoprint, microfilm or any other means without written permission from the author.

Distributed Data Stream Processing and Task Placement on Edge-Cloud Infrastructure

Gayashan Niroshana Amarasinghe

Supervisors: Prof. Shanika Karunasekera, Dr. Aaron Harwood, and Dr. Marcos Dias de Assunção

---

Abstract

Indubitable growth of smart and connected edge devices with substantial processing power has made ubiquitous computing possible. These edge devices either produce streams of information related to the environment in which they are deployed or the devices can be located in proximity to such information producers. Distributed Data Stream Processing is a programming paradigm that is introduced to process these event streams to acquire relevant insights in order to make informed decisions. While deploying data stream processing frameworks on distributed cloud infrastructure has been the convention, for latency critical real-time applications that rely on data streams produced outside the cloud on the edge devices, the communication overhead between the cloud and the edge is detrimental. The privacy concerns surrounding where the data streams are processed is also contributing to the move towards utilisation of the edge devices for processing user-specific data. The emergence of Edge Computing has helped to mitigate these challenges by enabling to execute processes on edge devices to utilise their unused potential. Distributed data stream processing that shares edge and cloud computing infrastructure is a nascent field which we believe to have many practical applications in the real world such as federated learning, augmented/virtual reality and healthcare applications.

In this thesis, we investigate novel modelling techniques and solutions for sharing the workload of distributed data stream processing applications that utilise edge and cloud computing infrastructure. The outcome of this study is a series of research works that emanates from a comprehensive model and a simulation framework developed using this model, which we utilise to develop workload sharing strategies that consider the intrinsic characteristics of data stream processing applications executed on edge and cloud resources.

First, we focus on developing a comprehensive model for representing the inherent characteristics of data stream processing applications such as the event generation rate and the distribution of even sizes at the sources, the selectivity and productivity distribution at the operators, placement of tasks onto the resources, and recording the metrics such as end-to-end latency, processing latency, networking latency and the power consumption.

We also incorporate the processing, networking, power consumption, and curating characteristics of edge and cloud computing infrastructure to the model from the perspective of data stream processing. Based on our model, we develop a simulation tool, which we call ECSNeT++, and verify its accuracy by comparing the latency and power consumption metrics acquired from the calibrated simulator and a real test-bed, both of which execute identical applications. We show that ECSNeT++ can model a real deployment, with proper calibration. With the public availability of ECSNeT++ as an open source software, and the verified accuracy of our results, ECSNeT++ can be used effectively for predicting the behaviour and performance of stream processing applications running on large scale, heterogeneous edge and cloud computing infrastructure.

Next, we investigate how to optimally share the application workload between the edge and cloud computing resources while upholding quality of service requirements. A typical data stream processing application is formed as a directed acyclic graph of tasks that consist of sources that generate events, operators that process incoming events and sinks that act as destinations for event streams. In order to share the workload of such an application, these tasks need to placed onto the available computing resources. To this end, we devise an optimisation framework, consisting of a constraint satisfaction formulation and a system model, that aims to minimise end-to-end latency through appropriate placement of tasks either on cloud or edge devices. We test our optimisation framework using ECSNeT++, with realistic topologies and calibration, and show that compared to edge-only and cloud-only placements, our framework is capable of achieving $8-14\%$ latency reduction and $14-15\%$ energy reduction when compared to the conventional cloud only placement, and $14-16\%$ latency reduction when compared to a naive edge only placement while also reducing the energy consumption per event by $1-5\%$.

Finally, in order to cater the multitude of applications that operate under dynamic conditions, we propose a semi-dynamic task switching methodology that can be applied to optimise end-to-end latency of the application. Here, we approach the task placement problem for changing environment conditions in two phases: in the first phase respective locally optimal task placements are acquired for discrete environment conditions which are then fed to the second phase, where the problem is modelled as an Infinite Horizon Markov Decision Process with discounted rewards. By solving this problem, an optimal policy can be obtained and we show that this optimal policy can improve the performance of distributed data stream processing applications when compared with a dynamic greedy task placement approach as well as static task placement. For real-world applications executed on ECSNeT++, our approach can improve the latency as much as $10-17\%$ on average when compared to a fully dynamic greedy approach.

Declaration of Authorship

This is to certify that,

- the thesis comprises only my original work towards the PhD,
- due acknowledgement has been made in the text to all other material used,
- the thesis is less than 100,000 words in length, exclusive of tables, maps, bibliographies and appendices.

Gayashan Niroshana Amarasinghe, January 2021

Preface

The research covered in this thesis has been carried out at The School of Computing and Information Systems, The University of Melbourne. The main contributions of this thesis are discussed in Chapters 3-5 and are based on the following publications where I was the primary author that contributed for more than 50% of the work:

- **Gayashan Amarasinghe**, Marcos Dias de Assunção, Aaron Harwood and Shanika Karunasekera, "A Data Stream Processing Optimisation Framework for Edge Computing Applications," *2018 IEEE 21st International Symposium on Real-Time Distributed Computing (ISORC)*, 2018, pp. 91–98.
- **Gayashan Amarasinghe**, Marcos Dias de Assunção, Aaron Harwood and Shanika Karunasekera, "ECSNeT++: A Simulator for Distributed Stream Processing on Edge and Cloud Environments," *Future Generation Computer Systems*, vol. 111, pp. 401–418, 2020.
- **Gayashan Amarasinghe**, Marcos Dias de Assunção, Aaron Harwood and Shanika Karunasekera, "A Two-Phase Semi-Dynamic Task Placement Strategy for Data Stream Processing on Edge Computing Infrastructure," *IEEE Transactions on Parallel and Distributed Systems.* (Under review)

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my principal supervisor, Professor Shanika Karunasekera. From the first point when she replied to my inquiry about research opportunities, to the present where I conclude my thesis, she has always remained approachable and directed me towards thoughtful resolutions whether the matter was related to my research or life as a whole. I would also like to thank my co-supervisor, Dr. Aaron Harwood who has always remained supportive throughout this research, and has provided his insightful remarks and careful analysis of the problems, which guided my thought process to develop solutions. I cherish the long discussions we had in his office, drawing solutions on the whiteboard, and imagining future directions. I was fortunate enough to have a third supervisor, Dr. Marcos Dias de Assunção who lived about 10 hours behind Melbourne, but was reachable with a short message. I thank him for always being there for me when I needed an advice and reaching out to check up on me frequently. I consider him as my mentor and hope that I could help others the way he has helped me. I am extremely grateful for all three of them, for helping me to grow as a junior researcher and for providing me with their invaluable time, advices, encouragements, criticism, and insights, all of which were indispensable to the completion of this thesis.

I would like to thank the chair of my PhD advisory panel, Professor Alistair Moffat for his guidance throughout my candidature, and showing me the importance of proper planning from early on. I also want to express my gratitude to Julie Ireland, Rhonda Smithies, and the whole administration team of the School of Computing and Information Systems and the Melbourne School of Engineering, for their continuous support in numerous matters. I am thankful for Dr. Maria Rodriguez Read for being a co-author of a contribution outside the scope of this thesis, and attending my completion seminar as an expert on the research area.

I acknowledge the financial support provided by the Melbourne Research Scholarship and the Elite PhD Scholars' Programme by The University of Melbourne, the University Grants Commission of Sri Lanka and the University of Moratuwa. I am grateful for the facilities and resources provided by the School of Computing and Information Systems to pursue my research.

I am thankful to all my friends, Sameera Kannangara, Sachini Jayasekara, Lakmal Muthugama, Sadegh Motellabi, Nicholas Akinyokun, Tabinda Sarwar, Damith Senanayake, Prashan Madumal, Soheila Ghane, Malinga Perera, and Dr. Xunyun Liu from the university for lending an ear when needed. I am glad that I had some of my childhood friends here to keep company, Charith and Maheeka, Maneesha and Melani,

Sandamal, Hirushan and Isuri, Harsha and Kavindya, and Rashmika and Achini. I thank them for making our life in Australia enjoyable.

Most importantly, I will not be here without the unconditional support from my family. I would like to thank my parents, Rasika and Wasantha Amarasinghe, who supported me from the very beginning and instilled in me the importance of education. Their immense sacrifices made me who I am. My brother, Kithmal Amarasinghe, was the strength behind my decision to leave for my studies, as I knew he would always be there to look after my parents. Special gratitude goes to his fiancé Thanushya as well. My extended family has provided me and my wife with strength and encouragement throughout this journey. My in-laws, Mollykanthi and Saman Weerasinghe, and Sasini and Chamath have always been there for us through thick and thin. I am glad that the love and comfort of family has fuelled this endeavour.

Highlight of this journey is the fortune of having my lovely wife, Induni, by my side. I am immensely grateful for her as she has been the source of strength for me throughout. I am thankful for her sacrifices as we share all the sweets and bitters of pursuing a PhD. Her unconditional love and constant encouragement has carried me through many difficulties in life. More than anyone else, she made this possible.

Lastly, I would like to thank all of you, whether explicitly mentioned or not, for all the tangible and intangible contributions throughout my journey as a graduate student.

Gayashan Amarasinghe
Melbourne, Australia
January 2021

Contents

Abstract ii

Declaration of Authorship iv

Preface v

Acknowledgements vi

List of Figures xiii

List of Tables xv

1 Introduction 1
- 1.1 Motivation . . . 4
- 1.2 Challenges in Distributed Stream Processing on the Edge and Cloud . . . 5
  - 1.2.1 Modelling the Characteristics of DSP Applications Executed on Edge-Cloud Computing Infrastructure . . . 5
  - 1.2.2 Designing an Environment for Conducting Comprehensive and Reproducible Experiments . . . 7
  - 1.2.3 Static Task Placement for DSP Applications with Static Conditions 8
  - 1.2.4 Dynamic and Reactive Task Placement in a Varying Application Environment . . . 8
- 1.3 Research Problems and Objectives . . . 9
- 1.4 Thesis Contributions . . . 10
- 1.5 Evaluation Methodologies . . . 11
- 1.6 Thesis Organisation . . . 12

2 Literature Review 14
- 2.1 Background . . . 16
  - 2.1.1 Evolution of Edge Computing and Related Infrastructure . . . 17
  - 2.1.2 Evolution of Data Stream Processing . . . 20
- 2.2 Taxonomy . . . 21
- 2.3 Application . . . 23
  - 2.3.1 Architecture . . . 23
    - 2.3.1.1 Composition . . . 24
    - 2.3.1.2 Interaction Type . . . 25
  - 2.3.2 Requirements . . . 26
    - 2.3.2.1 Resource Requirements . . . 26
    - 2.3.2.2 Service Requirements . . . 27

2.3.2.3 Policy Requirements . . . 27
2.3.3 Service Delivery Method . . . 28
2.3.3.1 Actuation . . . 28
2.3.3.2 Visualisation . . . 29
2.3.3.3 Storage . . . 29
2.3.3.4 Intermediate . . . 30
2.4 Infrastructure . . . 30
2.4.1 Computing Resource Types . . . 31
2.4.1.1 Cloud Computing . . . 31
2.4.1.2 Edge Computing . . . 32
2.4.1.3 Edge-Cloud Infrastructure . . . 32
2.4.2 Processing Platforms . . . 33
2.4.2.1 General Purpose Platforms . . . 33
2.4.2.2 Managed DSP Platforms . . . 33
2.5 Task Placement . . . 34
2.5.1 Modelling Strategies . . . 35
2.5.1.1 Constraint Satisfaction Problems . . . 36
2.5.1.2 Integer Programming . . . 36
2.5.1.3 Markov Decision Process . . . 38
2.5.1.4 Other . . . 39
2.5.2 Optimisation Objectives . . . 39
2.5.2.1 Time . . . 40
2.5.2.2 Resource Utilisation . . . 40
2.5.2.3 Energy/Power Consumption . . . 40
2.5.2.4 Cost . . . 40
2.5.2.5 Quality of Experience . . . 41
2.5.3 Dynamism in Task Placement . . . 41
2.5.3.1 Static Task Placement . . . 44
2.5.3.2 Dynamic Task Placement . . . 44
2.5.4 Placement Controller Positioning . . . 45
2.5.4.1 Centralised Placement Controller . . . 45
2.5.4.2 Distributed Placement Controller . . . 46
2.6 Evaluation Tools . . . 47
2.6.1 Physical Testbeds . . . 47
2.6.2 Simulators . . . 48
2.7 Summary . . . 50

3 A Model-based Simulator for Distributed Data Stream Processing 51
3.1 Introduction . . . 51
3.2 Background . . . 55
3.2.1 Characteristics of a Host Network . . . 55
3.2.2 Characteristics of a DSP Application . . . 56
3.3 ECSNeT++ Simulation Toolkit . . . 57
3.3.1 Host Network and Processing Nodes . . . 57
3.3.2 DSP Application . . . 59
3.3.2.1 Placement Plan . . . 60
3.3.2.2 Source . . . 60

3.3.2.3 Operator . . . 61
3.3.2.4 Sink . . . 61
3.3.3 Processing Model . . . 62
3.3.4 Power and Energy Model . . . 63
3.4 Experiments and Evaluation . . . 65
3.4.1 Application Topologies and Allocation Plans . . . 66
3.4.2 Measurements . . . 67
3.4.2.1 Delay Measurements . . . 69
3.4.2.2 Power and Energy Measurements . . . 69
3.4.3 Calibrating the Simulator . . . 70
3.4.4 Evaluation of Latencies . . . 71
3.4.4.1 Network Latency . . . 71
3.4.4.2 Processing Latency . . . 72
3.4.4.3 Total Latency . . . 73
3.4.5 Evaluation of Power Consumption . . . 73
3.4.6 Extending the Network Model with LTE Connectivity . . . 78
3.5 Comparison with Related Work . . . 80
3.6 Conclusion . . . 83

4 A Data Stream Processing Optimisation Framework for Static Task Placement 85
4.1 Introduction . . . 85
4.2 Optimisation Framework . . . 87
4.2.1 System Model . . . 87
4.2.2 Characteristics of a DSP application . . . 88
4.2.3 Constraint Satisfaction Problem . . . 88
4.2.3.1 Variables and Domains . . . 89
4.2.3.2 Residency Constraints . . . 90
4.2.3.3 Resource Constraints . . . 90
4.2.4 Optimisation Problem . . . 91
4.3 Simulation Design and Configuration . . . 92
4.3.1 Host Network and Processing Nodes . . . 92
4.3.2 DSP Applications . . . 93
4.3.3 Processing Model . . . 94
4.3.4 Power Model . . . 94
4.4 Evaluation . . . 95
4.4.1 Task Placement Plans . . . 95
4.4.2 Results . . . 96
4.4.2.1 Performance of the Micro-benchmarks . . . 96
4.4.2.2 Performance of the ETL+STATS application . . . 98
4.5 Comparison with Related Work . . . 100
4.5.1 Task Placement Strategies . . . 100
4.5.2 Distributed Stream Processing on the Edge . . . 102
4.6 Conclusion . . . 102

5 A Semi-Dynamic Task Placement Strategy for Data Stream Processing 104
5.1 Introduction . . . 104

5.2 Background . . . 106
5.3 Problem Formulation . . . 108
5.3.1 States . . . 108
5.3.2 Actions . . . 110
5.3.3 Discrete Time Sampling . . . 110
5.3.4 Transition Probabilities . . . 110
5.3.5 Reward Function . . . 111
5.3.6 Policies . . . 114
5.3.7 Optimality Criteria . . . 114
5.3.8 Value Function . . . 114
5.4 Performance Evaluation . . . 115
5.4.1 Policies . . . 116
5.4.1.1 Two-phase Semi-dynamic Task Placement strategy (TSTP) Policy . . . 116
5.4.1.2 Greedy Policy . . . 117
5.4.1.3 Resource-aware Placement Policy (RA) . . . 117
5.4.2 Applications . . . 117
5.4.2.1 Simple DSP application . . . 118
5.4.2.2 STATS Application . . . 119
5.4.2.3 ETL+STATS Application . . . 120
5.4.3 Performance Evaluation Results . . . 122
5.4.4 Mitigating State Explosion in TSTP . . . 124
5.4.5 Deploying Multiple Applications with TSTP . . . 127
5.5 Comparison with Related Work . . . 127
5.6 Conclusion . . . 128

6 Conclusions and Future Directions **129**
6.1 Summary and Conclusions . . . 130
6.1.1 A Model-based Simulator for Distributed Data Stream Processing 131
6.1.2 An Optimisation Framework for Task Placement for Applications Operating Under Static Conditions . . . 132
6.1.3 A Semi-Dynamic Task Placement Strategy for Applications Operating Under Dynamic Conditions . . . 133
6.2 Future Directions . . . 134
6.2.1 Optimising Task Placement for Multi-tenant DSP Applications . . . 134
6.2.2 Extending Two-phase Semi-dynamic Task Placement as a Model-free Markov Decision Process . . . 135
6.2.3 Handling Fault Tolerance . . . 135
6.2.4 Implementing Support for Privacy and Security Policies . . . 136
6.3 Final Remarks . . . 137

Bibliography **138**

List of Figures

1.1 A complex data stream processing application with a source, 16 operators and 2 sinks . . . 2
1.2 Organisation of the thesis. . . . 13

2.1 Overview of cloud-centric and edge-cloud task placement for distributed data stream processing applications . . . 16
2.2 Timeline of the evolution of the data stream processing and the evolution of cloud and edge computing paradigms . . . 18
2.3 The high-level taxonomy of data stream processing and task placement on edge cloud infrastructure . . . 22
2.4 The taxonomy of data stream processing applications . . . 23
2.5 The taxonomy of the infrastructure for data stream processing . . . 30
2.6 The taxonomy of formulating and solving the data stream processing task placement problem . . . 35

3.1 Components and characteristics of a DSP application . . . 56
3.2 The high level architecture of ECSNeT++ . . . 58
3.3 Multi-core multi-threaded processing model of ECSNeT++ . . . 62
3.4 Experimental set-up used for simulator calibration and verification . . . 66
3.5 Simplified ETL application topology from the RIoTBench suite . . . 66
3.6 Simplified STATS application topology from the RIoTBench suite . . . 67
3.7 Possible placement plans for the simplified ETL application topology . . . 68
3.8 Power measurement set-up for the Raspberry Pi 3 devices . . . 69
3.9 Comparison of latency on experiments and simulation for the ETL application at 10 events per second . . . 74
3.10 Comparison of latency on experiments and simulation for the ETL application at 100 events per second . . . 75
3.11 Comparison of latency on experiments and simulation for the STATS application at 5 events per second . . . 76
3.12 Comparison of latency on experiments and simulation for the STATS application at 10 events per second . . . 77
3.13 Comparison of average power consumption on experiments and simulation for ETL application at 10 events per second . . . 78
3.14 Comparison of average power consumption on experiments and simulation for ETL application at 100 events per second . . . 79
3.15 Comparison of average power consumption on experiments and simulation for STATS application at 5 events per second . . . 79
3.16 Comparison of average power consumption on experiments and simulation for STATS application at 10 events per second . . . 80

3.17 Simulated LTE environment with 5 LTE enabled mobile phones connected using LTE Radio Access Network . . . 81
3.18 Latency of the ETL application at 10 events per second on the simulated LTE network . . . 82

4.1 Host network with edge and cloud infrastructure . . . 93
4.2 Micro-benchmark topologies . . . 94
4.3 Topology and task placement of a selected synthetic micro-benchmark and the ETL+STATS application . . . 96
4.4 Mean end-to-end latency of micro-benchmarks . . . 98
4.5 Mean end-to-end latency of ETL+STATS topology on PiB devices . . . 99
4.6 Mean end-to-end latency of ETL+STATS topology on Pi3B devices . . . 99

5.1 Simple data stream processing application . . . 107
5.2 Partial state transition diagram for all the actions available from state $\langle low, m_0 \rangle$ . . . 111
5.3 Distribution of the latency reward for the Simple application . . . 113
5.4 Simulated edge-cloud infrastructure used for executing the applications . . . 116
5.5 STATS application topology . . . 119
5.6 ETL+STATS application topology . . . 122
5.7 Time-lines of per-event latency measurements for processing 10000 events of the STATS application . . . 124
5.8 Summarised mean per-event latency for processing 10000 events of the STATS application . . . 125
5.9 Summarised mean per-event latency of processing 50000 events of the Simple application . . . 125
5.10 Summarised mean per-event latency for processing 10000 events of the ETL+STATS application . . . 126

List of Tables

2.1 Mono-objective optimisations presented by the state-of-the-art in DSP on edge and cloud environments . . . 42
2.2 Multi-objective optimisations presented by the state-of-the-art in DSP on edge and cloud environments . . . 43

3.1 Comparison of ECSNeT++ simulation toolkit against state-of-the-art . . 54
3.2 Notations used in the modelling of ECSNeT++ . . . 59
3.3 Specification of the edge and cloud devices used in experimental set-up . . 65
3.4 Characteristics of the operators of the ETL and STATS applications . . . 67
3.5 Network configuration used in the experimental set-up and the simulation 71
3.6 Measured average power consumption values used for configuring Raspberry Pi 3 Model B devices in the simulation . . . 71

4.1 Configurations for the edge and cloud devices . . . 93
4.2 Power consumption configuration of Raspberry Pi devices . . . 95
4.3 Parameters used for simulation . . . 95
4.4 Per-event mean energy consumption of micro-benchmark topologies . . . 100

5.1 Notations used in the the formulation of TSTP . . . 109
5.2 Acquired placement plans for the simple application and their measured latency values . . . 119
5.3 Acquired placement plans for STATS application and their measured latency values . . . 120
5.4 Acquired placement plans for ETL+STATS application and their measured latency values . . . 121
5.5 Properties of the operators of the applications used for evaluation of TSTP 122

Chapter 1

Introduction

Information and communication technologies have revolutionised the way we solve problems. Many day to day problems are modelled using computer based systems and software based solutions are being used to solve these problems. Internet and the World Wide Web have become essential commodities of the day to day life. As a result, we have reached the present Big Data Era with Internet-scale web applications and trillions of interconnected devices where data is being generated with an exponential growth rate.

The data often can be represented as a series of events which is commonly referred to as a data stream. With the comprehensive understanding of the specific characteristics of these data streams, Data Stream Processing programming paradigm is enacted for processing them to extract valuable information. Data Stream Processing is mainly concerned about processing an unbounded stream of data events that is often high in volume, velocity and variety. The event streams are generated by many heterogeneous sources depending on the application and the environment. For example, social media platforms such as Twitter or reddit produce diverse and large-scale streams of events continuously. Mobile phones, wearable devices, connected vehicles, and smart devices such as Internet of Things (IoT) produce unbounded streams of events about users, current operating state of devices or their surrounding environment. These events need to be processed to acquire knowledge about the application, user or the environment in order to make decisions, which is the requirement satisfied by Data Stream Processing. The definition for Data Stream Processing has evolved to Distributed Data Stream Processing since these data streams are being generated and processed by distributed systems. Without loss of generalisation, the abbreviation DSP[1] will be used to refer to Distributed Data Stream Processing in its broadest sense from here onwards.

[1]Should the need arise to refer, any signal processing related terminology such as digital signal processing, will not be abbreviated in this thesis to avoid ambiguity.

A DSP application is formed as a directed acyclic graph that consists of 3 types of components as vertices, known commonly as tasks, which are *sources* that generate events, *operators* that process events, and *sinks* that act as destinations for results. Each task is a decoupled, executable component with a specific functionality. The connection between these tasks is known as a *data flow*. A DSP application is organised as a topology of these tasks connected with their data flow of events which are generated by the sources. Every operator manipulates the incoming data flow and generates an output data flow that is sent to one or more of the down stream tasks until it reaches a sink. A DSP application topology with a single source, 16 operators and 2 sinks is illustrated in Figure 1.1.

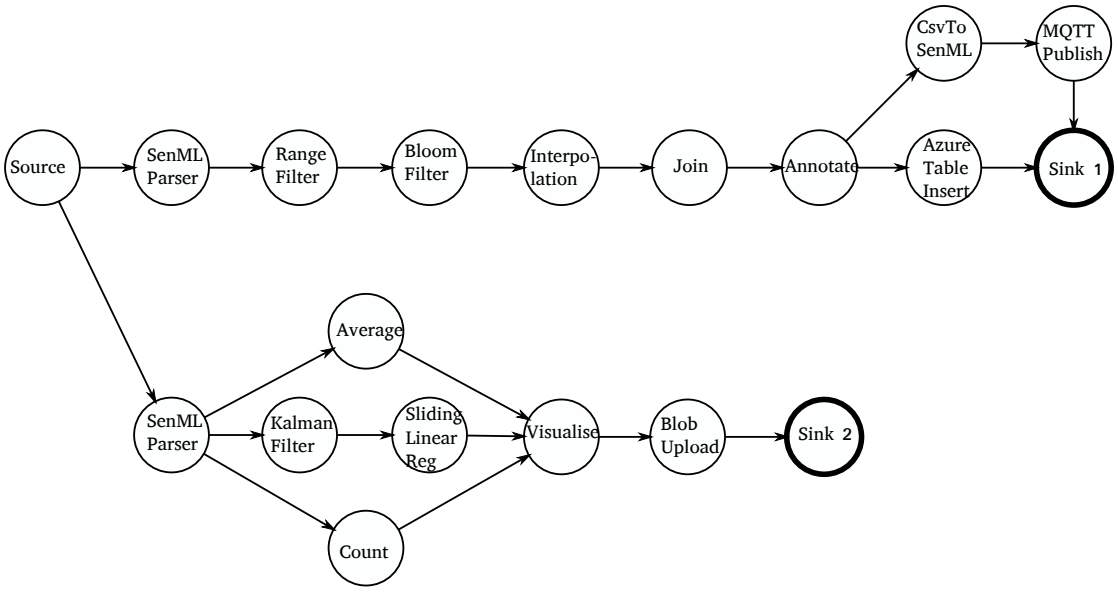


FIGURE 1.1: A complex DSP application with a source, 16 operators and 2 sinks from the RIoTBench Suite [1]

When a DSP application is to be executed, its tasks need to be executed on available computing resources and the data flow occurs through the network between the devices or via local memory hierarchy within each device. Therefore, in order to execute a DSP application, each task has to be mapped to one or more of the available computing resources. While a variety of terms are used for naming the mapping of individual tasks to available resources, this thesis will use the term *Task Placement* for this purpose.

One of the main requirements of many DSP applications is to process the event streams in real-time, i.e. with low latency, among other requirements such as achieving high throughput or low power consumption. Many DSP applications require results to be produced in near real time, such as processing a video stream of an augmented reality application to identify components of the surrounding environment, a healthcare

application that monitors a stream of events from monitoring devices worn by a critical patient, or an automated motor vehicle that analyses a stream of events about the surrounding environment to determine the safe speed to follow a car in front.

Many data stream processing engines (DSPEs) are being developed, and used in the industry and the academia to execute DSP applications with a task placement, such as Apache Flink [2] and Apache Beam [3] which are used and supported by Google, and Apache Storm [4] and Apache Heron [5] which are used and supported by Twitter. However, these frameworks are designed to be deployed in the cloud and as a result, a common pattern is to produce the event streams in sources located outside the cloud as mentioned earlier which are then pushed to the cloud to be processed. This is not an ideal arrangement due to multiple reasons. The communication overhead of a medium such as a wireless connection that connects an event source and the cloud may result in high latency which hinders the latency requirements of DSP. Data streams are heterogeneous which results in requiring bespoke solutions for each type of stream. In addition, due to the privacy concerns, where data is processed and stored needs to be carefully considered. Therefore, producing events outside the cloud and processing them in the cloud has become an infeasible arrangement for many latency critical applications.

With the rapid growth of the connected devices, Edge Computing is emerging as a complimenting processing platform to cloud computing. Edge computing is concerned about executing applications in devices logically located outside the cloud in varying depths, starting from outside the cloud network perimeter to the end user. Mobile phones, personal computers, wearable devices, single board computers such as Raspberry Pis[2], mobile cellular base stations, IoT devices, smart vehicles, smart routers and switches, or any connected device with some processing power, can be considered as edge devices. The goal of edge computing is to bring the processing closer to the data sources.

Due to advancements in mobile processor technologies, edge computing resources are gaining substantial processing power which is often underutilised. These devices are often owned by the users, and as a result privacy concerns can be mitigated by utilising the processing and storage capacity of the devices for processing personal data. In addition, these devices are located closer to the event sources or frequently the devices themselves act as the event sources. The edge devices are connected through wired or wireless communication media and are able to communicate with the cloud computing resources. Therefore, considering these resources for DSP task placement provides an opportunity to both utilise these underutilised resources and to overcome the challenges such as lower latency and improved privacy at the same time. As a result, this thesis

[2]Raspberry Pis are commercially available, small profile, single board computers with a variety of configurations such as single core processor Pi Zero devices to quad-core Pi 4 Model B devices. https://www.raspberrypi.org/products/

mainly focuses on utilising edge computing resources along with the cloud computing resources to share the data stream processing workload to produce results with low latency.

1.1 Motivation

Edge Computing is attracting the attention of the industry due to the potential market share it represents and we can clearly observe solutions being introduced such as Google's implementation of artificial intelligence at the edge [6], Amazon's AWS IoT for the Edge and the AWS IoT Greengrass platform [7] that allows applications to run on local devices, Microsoft's Azure IoT Edge [8] that allows to run business logic on edge devices via standard containers, in-device video analysis devices [9][10][11], home automation devices [12], and industrial automation applications [13]. Therefore, we can expect a surge in the utilisation of these edge devices for various applications.

DSP has a wide range of applications in the real world. These applications can range from a simple application, that monitors a water level sensor to control a water pump, to a complex application, that monitors the stock market to make predictions for trading. The set of DSP applications that has a potential to utilise edge computing resources is expanding through various domains as well.

In the healthcare domain, a critical diabetic patient's ongoing health can be monitored via a set of wearable devices and an application can be used to adjust the insulin level based on the frequently measured blood sugar level and the history of the patient. Instead of conducting the processing at the cloud, the patient's mobile device or a similar smart device can be used locally to process this stream of events from the monitoring devices and to make the appropriate decisions. This solution can both provide results with low latency as well as protect the privacy of patient's data.

Federated learning is another application which can benefit from DSP paradigm and utilising edge resources. With federated learning, the goal is to train the machine learning models on the devices that generate events and to aggregate the models in the cloud in a privacy preserving manner where this process can be modelled as a DSP application topology and the application workload can be shared with the edge devices that generate the events. Gboard, the virtual keyboard implemented by Google, utilises federated learning to train a user-specific model on user devices using the streams of input events from the user, which only periodically gets synchronised with a global training model in the cloud, to provide next-word prediction [14].

Augmented reality and virtual reality applications present another opportunity to utilise DSP and edge computing. These applications process a continuous stream of events that includes a video feed and events from various sensors related to the surrounding environment. This stream needs to be processed in real time in order to provide meaningful information and constant feedback to the application user, and as a result the edge devices can be utilised to improve the quality of service provided, instead of completely relying on the cloud computing resources.

Applications can have a real impact on the world and solve real problems, and the approaches we propose can improve the experience provided to the end user which motivated us to conduct this study. We have identified common types of applications which are 1) extract, transform and load applications where the input stream is filtered, pre-processed and loaded to storage or another application, and 2) statistical analysis applications that analyse an input stream to provide statistical data or summarisations of the inputs. Therefore, we focus on these types of applications in our research.

We also found that task placement has been widely studied in the literature such as [15][16][17][18][19][20] and [21] but found that the challenges imposed by edge computing resources on running DSP applications are not explored sufficiently, which motivated us to propose solutions to address some of these challenges that are vital for optimal utilisation of edge and cloud resources.

1.2 Challenges in Distributed Stream Processing on the Edge and Cloud

Sharing the DSP workload with edge and cloud computing resources while maintaining the required Quality of Service guarantees has brought many challenges.

1.2.1 Modelling the Characteristics of DSP Applications Executed on Edge-Cloud Computing Infrastructure

Before we analyse how to optimise the execution of DSP applications on edge and cloud infrastructure, we need to first understand the characteristics of them. An intricate model of the application and the resources can be beneficial to this end as a model allows to manipulate the various parameters emerging from these characteristics and to observe the corresponding behaviour before making concrete decisions.

The model of a DSP application needs to consider the inherent requirements of every component of the application. Sources of a DSP application produce event streams and

as a result are concerned about the distribution of the event generation rate and the distribution of the size of the events generated by the source. For instance, a DSP application that tracks human behaviour, such as a fitness tracker may have a bimodal event generation rate distribution where high event generation rates are expected in mornings and evenings. Each operator of a DSP application manipulates the incoming event streams to produce outgoing event streams based on the application logic. Therefore, operators need to be able to clearly define the selectivity and productivity ratio distributions that decide the ratio between the number of events emitted to the number of events ingested and the ratio between the size of an emitted event to the size of an ingested event respectively. For example, a windowing operator that takes average of a stream of the last 100 temperature readings may emit an event for every 100 incoming events. Sinks act as destinations for the event streams and also record measurements such as the end-to-end latency of an event, per event processing and network transmission time, and per event energy consumption. In addition, each task has a per event processing time that is associated with it. The model should also facilitate the task placement onto the resources as when the tasks are placed on one or more resources, the event streams from one task to the another needs to be configured according to the application topology.

The model of the infrastructure used to execute the DSP application needs to be able to capture the composition of the resources, the processing capabilities of the resources such as multi-core, multi-threaded architectures, network connectivity options such as wired and wireless, and power consumption characteristics. For example, edge resources have many different CPU configurations that result in different instruction sets being used to execute the applications, and as a result application compatibility is a main concern across different edge computing resources. The network connectivity support of edge resources is varying from wired Ethernet to wireless technologies such as Wi-Fi or Zigbee to cellular technologies such as LTE or 5G. The power consumption characteristics of edge resources vary from devices continuously powered via the power grid to battery powered devices to solar powered devices. The model should take into account that multiple heterogeneous resources need to be interconnected in order to set-up the edge-cloud infrastructure. In addition, it is also important to model characteristics such as the mobility of the devices.

Lastly, the model should also allow for the recording of metrics that are vital to evaluate the performance of the DSP applications. End-to-end latency as well as the breakdown of processing and transmission latencies are needed to compare the effectiveness task placement policies as well as the approaches used to produce the policies. Similarly, the power or energy consumption needs to be able to be recorded using the model while recording the contributions by processing, networking and idle energy consumption.

1.2.2 Designing an Environment for Conducting Comprehensive and Reproducible Experiments

The objective of this thesis is to share the data stream processing workload with the edge and cloud computing resources. In order to devise effective methodologies to achieve this objective, we need a realistic environment to conduct experiments to evaluate the effectiveness of the proposed methodologies. The realistic test environment should consist of edge and cloud computing devices, wired and wireless networking options to connect the devices, and the ability to deploy DSP applications to collect performance metrics, all of which are crucial for evaluating the effectiveness of workload sharing methodologies. The edge devices are heterogeneous with varying resource capacities, framework and protocol support, operating systems, instruction sets, networking capabilities, and power consumption characteristics. The networking infrastructure can be either wired or wireless with numerous standards and protocols depending on the requirements of the applications. The scale of the test environment should be able to capture the magnitude of the edge computing infrastructure where the number of connected devices for a particular application can range from a handful of devices to millions of devices. The locality of the resources should be able to represent the devices deployed in a single household to a network of devices scattered over multiple continents. The test environment should be able to build a set up for testing with a mixture of these heterogeneous devices and support executing data stream processing applications with their own inherent characteristics on the devices with appropriate task placement. While representing these characteristics, the test environment should also produce reproducible results for verification.

A realistic testbed with physical devices is capable of catering to many of these requirements. However, due to the heterogeneity of the required resources and the connectivity options, it could become technically and financially overwhelming to purchase, configure, and connect variety of devices to form a comprehensive environment. Therefore, it is not feasible to develop such a comprehensive test environment that can also execute reproducible experiments, which led to the option of simulating the test environment. A simulator designed to achieve this purpose, should be able to cater all of the requirements and also be able to be verified against a real test bed sufficiently, to be effective. Therefore, the simulator needs to be designed using a model that captures the characteristic of DSP applications, and the edge and cloud infrastructure.

1.2.3 Static Task Placement for DSP Applications with Static Conditions

When the behaviour of the application or the environment in which it operates largely remains static during the lifespan of the application, instead of a trivial task placement, a methodology to acquire an optimal or near-optimal task placement plan that achieves an optimisation goal such as the minimisation of the end-to-end latency is required. This task placement plan should be acquired by analysing the application resource requirements and the availability of the resources such as CPU, memory and network bandwidth. In addition, residency concerns of the application tasks need to be considered when placing a task onto one or more resources. For instance, due to privacy concerns or hardware constraints, a task may be required to be placed on a specific type of a resource. As the applications and the environment become complex, and the amount of tasks and the available resources grow in size and variety, a brute force exhaustive analysis of all the possible task placement plans becomes intractable. On the other hand, due to large number of parameters that can affect the decision, this problem is proven to be a NP-hard problem and hence reduction of the search space is vital as well so that the problem becomes tractable.

1.2.4 Dynamic and Reactive Task Placement in a Varying Application Environment

When the application and the execution environment have a frequently changing behaviour, the task placement also has to be dynamic and reactive in order to provide the desired quality of service. However, frequent changes to the task placement plan has lasting consequences. Continuously monitoring and solving the problem to acquire optimal placement plans require substantial processing power which can be otherwise utilised. In addition, changing the task placement plan frequently is detrimental to the end-to-end latency of the overall execution of the application. For instance, one approach to change the current task placement is to pause the event stream, deploy the new placement plan, and replay the paused stream until it catches up with the real-time stream. During the deployment phase of the new plan, the DSP application is unavailable and as a result the subsequent phase could overload the system due the streams that are buffered. All these steps incur a time penalty and if conducted frequently, they could affect the latency guarantees. Furthermore, both processing for acquiring optimal placement plans and switching placement plans incur an energy penalty as well. Intractability of the problem with the growth of the application and the environment also affects the decision to acquire optimal placement plans. As a result, it is necessary to develop a

methodology to both make sure that the search space of the problem is reduced as well as to develop a framework that mitigates costs the frequent task switching has on the quality of service provided to the end users.

1.3 Research Problems and Objectives

This thesis focuses on overcoming the challenges of robust work load sharing of DSP applications on to edge and cloud computing resources and maintaining guaranteed latency requirements by identifying and investigating the following research problems.

- **How to model a DSP application and the underlying environment conditions to accurately represent their inherent characteristics?** In order to clearly understand the dynamics between the DSP application and the resources available to execute the application, a comprehensive system model is needed. The model needs not only to consider homogeneous resources, but needs to take into account the resource heterogeneity and define the characteristics of a DSP application accurately.

- **How to design a simulation tool to accurately represent the heterogeneity of the edge and cloud computing resources and the networks for executing DSP applications?** Due to the heterogeneity of the edge and cloud computing resources, setting up a real test environment that can run reproducible experiments with multiple configurations of devices has proven to be an expensive effort both financially and technically, and as a result, simulation has emerged as a viable alternative. However, the existing simulation tools lack the ability to fine tune the configuration parameters of DSP applications in order to manage the task placement. In addition, we found that the existing simulators are not properly verified against a real test environment. Since simulation without a sense of the accuracy of acquired results is inefficacious, a verified simulation tool with the ability to mimic various, large-scale device and network configurations, and configure the components of DSP applications as required is needed in order to strengthen the ongoing exploration efforts of utilising these resources.

- **How to optimally deploy the tasks of a DSP application on to the resources with fixed environmental and application conditions?** As the brute force method of optimal task placement for large scale applications with many tasks is infeasible, a methodology is needed which can be applied to any DSP application given a set of heterogeneous resources with the assumption that

application conditions remain static for the lifespan of the application. The expected methodology should devise optimal or near-optimal task placement plans by analysing the requirements of the application and the availability of the resources in applications and environments while optimising for a one or more well defined metric. Heuristics may also need to be applied here in order to reduce the search space to ensure that the solution remains tractable.

- **How to model dynamic and reactive switching between different task placements, and to execute the application with varying environmental and application conditions?** When the application requirements or the environment conditions are dynamic, the corresponding workload sharing placement plan also needs to be adjusted. However, dynamically switching between all the possible placement plans is not a viable alternative due to the high processing cost as well as the impact on the low latency requirement. As a result, a methodology is required that can overcome these processing and migration overheads and to acquire a reactive switching policy that optimises the latency requirements of the application.

1.4 Thesis Contributions

To solve these research problems, we summarise the major contributions made by this thesis below.

1. A taxonomy and a survey of edge computing paradigm and the state-of-the-art methodologies for executing DSP applications on edge and cloud computing resources.
2. A simulation tool named ECSNeT++ built on our modelling of executing DSP applications on edge and cloud computing resources.
    - A methodology for modelling the characteristics of DSP applications, task placement, and edge and cloud computing resources.
    - A public, open-source, and extendible simulation environment designed from this model to execute reproducible experiments.
    - Validation of the simulator by comparing the latency and power consumption measurements with an identical realistic test bed by executing similar DSP applications.
3. An optimisation framework for static placement of tasks on edge and cloud computing resources under fixed conditions.

- A mathematical model is formulated as a constraint satisfaction problem to model executing a DSP application topology on edge and cloud computing infrastructure.
- A framework is developed that captures the resource and residency constraints of the tasks of a DSP application.
- A constraint optimisation problem is formed for optimising the task placement problem for low latency.

4. A semi-dynamic and reactive task placement policy for handling varying conditions of the application and the resource environment.

   - A task placement framework formed to consider varying application conditions of the application and the execution environment modelled as a two-phase methodology based on a Markov Decision Process.
   - A methodology to reduce the state space of the Markov Decision Process by only considering locally optimal placement plans to bound the computational complexity to identified discrete application conditions.
   - A configurable reward function that captures the current state of the task placement and the next possible placements to achieve an optimal task placement with respect to the application conditions.
   - A strategy to switch between pre-deployed task placement plans semi-dynamically by considering the long term discounted reward.

1.5 Evaluation Methodologies

The solutions proposed in this thesis are evaluated using three synthetic DSP applications as well as two real world DSP applications. The three synthetic applications were developed to display three different application topologies, linear, diamond and a tree topology. The two real applications, ETL and STATS, are acquired from the RIoTBench suite [1]. Parsing, filtering, statistical and predictive analytics, pattern detection, visual analytics and IO operations are identified by Shukla, Chaturvedi and Simmhan [1] as key tasks performed in edge computing applications. Out of the two real applications that are considered in this thesis, the ETL application is used for extracting, transforming, and loading of sensor data into a database after conducting some pre-processing on the data whereas the STATS application is used for visualising and statistical aggregation of the data stream in order to acquire a high level perspective of the data. In some experiments, we consider the composition of ETL and STATS applications which is referred as the ETL+STATS application.

The simulation tool, ECSNeT++, is verified using a real world Edge-Cloud environment that consists of 2 Raspberry Pi 3 Model B devices acting as edge devices that interconnect via separate access points and then connects to a single cloud node via a router. Same network architecture and device configurations are created on the simulator which was also calibrated with the parameter values from the real set up, and the applications were executed on both environment for verifying the results acquired from the simulator. Due to the lack of access to physical edge computing resources, we utilise the calibrated and verified ECSNeT++ simulator for executing the rest of the experiments in this thesis.

End-to-end latency, processing latency, network latency, and the power and energy consumption at the edge devices are considered as the metrics used to evaluate the accuracy of the simulator as well as the performance of the proposed optimisation frameworks. Each optimisation framework is studied using the following methodology of 1) system modelling using mathematical tools, 2) problem formulation and solving, 3) implementation of the solution on ECSNeT++ and 4) performance evaluation and numerical analysis of results acquired by executing the synthetic or real applications to prove the effectiveness of the proposed frameworks as well as the simulator itself.

1.6 Thesis Organisation

The organisation of the chapters in this thesis is illustrated in Figure 1.2. Chapter 2 discusses the taxonomy and survey of the state-of-the-art of distributed data stream processing on edge and cloud computing systems. Chapter 3 covers the open source simulation environment that we have developed in order to execute the DSP applications on edge and cloud resources. Chapters 4 and 5 focus on two types of task placement strategies: static and dynamic task placement. Chapter 6 concludes the thesis with our findings as well as directions for future studies in this domain. The following core chapters of this thesis are mainly derived from the conference and journal publications completed during my PhD candidature.

- Chapter 3 proposes a simulation environment ECSNeT++ which we have developed to execute DSP applications on edge and cloud resources with complex networking, execution and power consumption models with the ability to calibrate and configure according to a real world environment. We have also verified this simulator against a real set up, and made it open source and publicly available for further use and extension. The chapter is derived from the following publication:
  - **Gayashan Amarasinghe**, Marcos Dias de Assunção, Aaron Harwood, and Shanika Karunasekera, "ECSNeT++ : A simulator for distributed stream

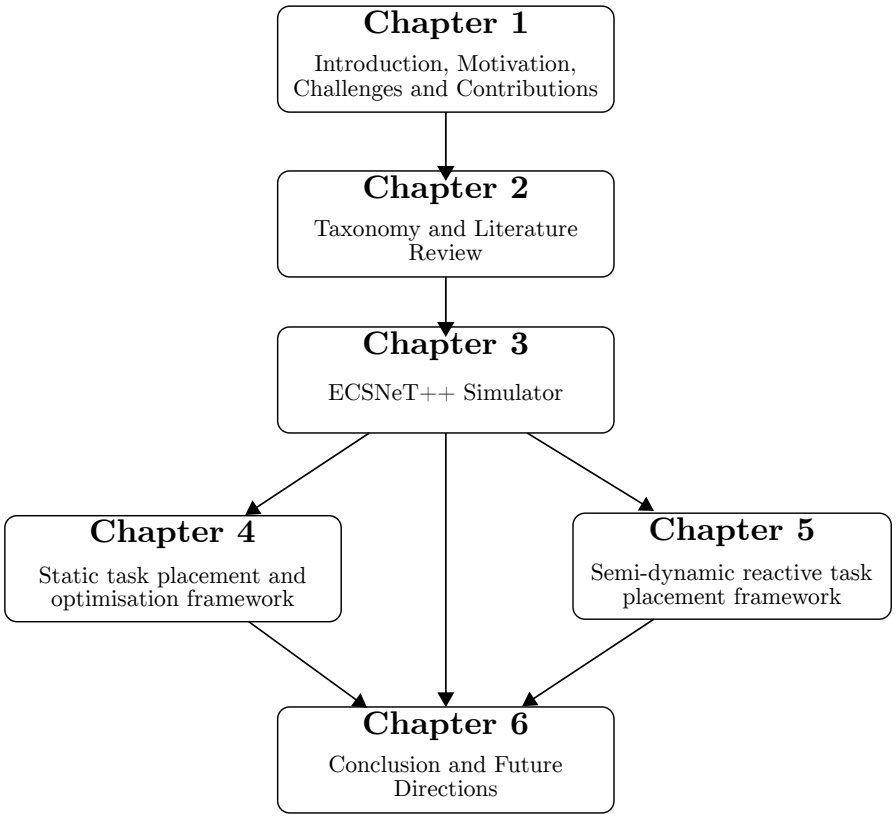


FIGURE 1.2: Organisation of the thesis.

processing on edge and cloud environments," *Future Generation Computer Systems*, vol. 111, pp. 401–418, 2020.

- Chapter 4 introduces a constraint satisfaction based mathematical model for static DSP task offloading on to edge and cloud resources. This chapter is derived from the following publication:
  - **Gayashan Amarasinghe**, Marcos D. de Assunção, Aaron Harwood, and Shanika Karunasekera, "A Data Stream Processing Optimisation Framework for Edge Computing Applications," *2018 IEEE 21st International Symposium on Real-Time Distributed Computing (ISORC)*, Singapore, 2018, pp. 91-98.
- Chapter 5 presents a semi-dynamic reactive task placement framework based on a model-based Markov Decision Process.
  - **Gayashan Amarasinghe**, Marcos D. de Assunção, Aaron Harwood, and Shanika Karunasekera, "A Two-Phase Semi-Dynamic Task Placement Strategy for Data Stream Processing on Edge Computing Infrastructure," *IEEE Transactions on Parallel and Distributed Systems* (Under review).
- Chapter 6 concludes the thesis by summarising our key findings and providing directions for future research.

Chapter 2

Literature Review

Novel developments in the processor, networking and battery technologies in the last two decades have resulted in the increasing use of smart devices in our day to day lives as well as in the industrial applications, as these devices have become ubiquitous beyond question. With the rapid deployment of such smart devices, the amount of information produced has also increased exponentially, leading to the continuous generation of unbounded streams of data events. These event streams need to be processed in a timely manner in order to make the decisions on controlling or monitoring the environment in which the devices are deployed. The streams can be processed using either batch processing or stream processing, whereas batch processing relates to a store-first and process-later model, stream processing is a process-on-arrival model. With stream processing, which is a paradigm for in-memory real-time processing of streams of events, the incoming events are handled immediately, and results are made available incrementally. By optimising the processing done to each event, the end-to-end latency of each result can also be reduced for applications that require real-time processing.

A complete stream processing system consists of a stream processing application, computing and networking infrastructure, and other auxiliary services. More often than not, the stream processing application and the supporting services are managed by a DSP Engine (DSPE) which is also often deployed on a distributed infrastructure. A modern DSP application is a distributed collection of tasks and dataflows, that can be arranged as a graph, where the vertices represent the tasks and the edges represent the dataflow between tasks. The infrastructure available to execute the application, consists of a network of connected cloud or edge computing nodes. Hence, in order to execute the application, each task needs to be mapped to a computing node, deployed on to the computing nodes, and routed accordingly. Mapping each task to a computing resource is referred to as Task Placement. If a DSPE is utilised, then it manages the deployment

of tasks onto computing resources according to the task placement policy and conducts the required housekeeping work such as routing event streams, collecting metrics of the application, monitoring the load on the nodes, and tolerating failures if needed.

The DSP paradigm is often designed to be deployed on readily available, robust, and scalable cloud computing instances. However, with the emergence of IoT and numerous other connected smart devices, the number of events that are generated outside the cloud has grown exponentially. As illustrated in Figure 2.1, with the cloud-centric DSP, the general approach is to submit these events, generated outside the cloud, to the cloud for processing. Due to the communication overhead of different communication media that connect these event sources, processing at the cloud, i.e., the traditional cloud-centric task placement, can become infeasible for some applications. As a result, Edge Computing paradigm has been introduced, which focuses on bringing the task processing closer to the event sources, onto the computing enabled devices, located outside the cloud and at the edge of the network, which can even be the devices themselves that generate events. However, some edge computing devices have its own disadvantages such as being battery powered thus prone to failures, being connected over wireless media and being relatively underpowered when compared to the cloud. As a result, a trade-off needs to be made when placing tasks onto the cloud and the edge to meet the requirements of the application owners, and to optimise the metrics such as the end-to-end latency and power consumption. While edge computing has been a relatively new paradigm, similar concepts have been considered since the inception of parallel computing. Nevertheless, we have identified that focusing on the inherent characteristics of Distributed Data Stream Processing and the characteristics of edge computing are crucial for identifying solutions for the task placement problem for DSP applications running on edge-cloud environments. For example, the requirement for real-time computing, the distributed architecture and graph based representation, and being in close proximity to sources with enough processing power to reduce the amount of data transferred to the cloud can be exploited. In order to explore solutions to this problem, we have conducted an exhaustive analysis on the state-of-the-art and make the following contributions in this chapter:

- Introduce the main concepts and approaches in data stream processing on edge-cloud resources.
- Present a comprehensive taxonomy for the state-of-the-art based on the characteristics and relevancy of each work for data stream processing on edge-cloud resources.

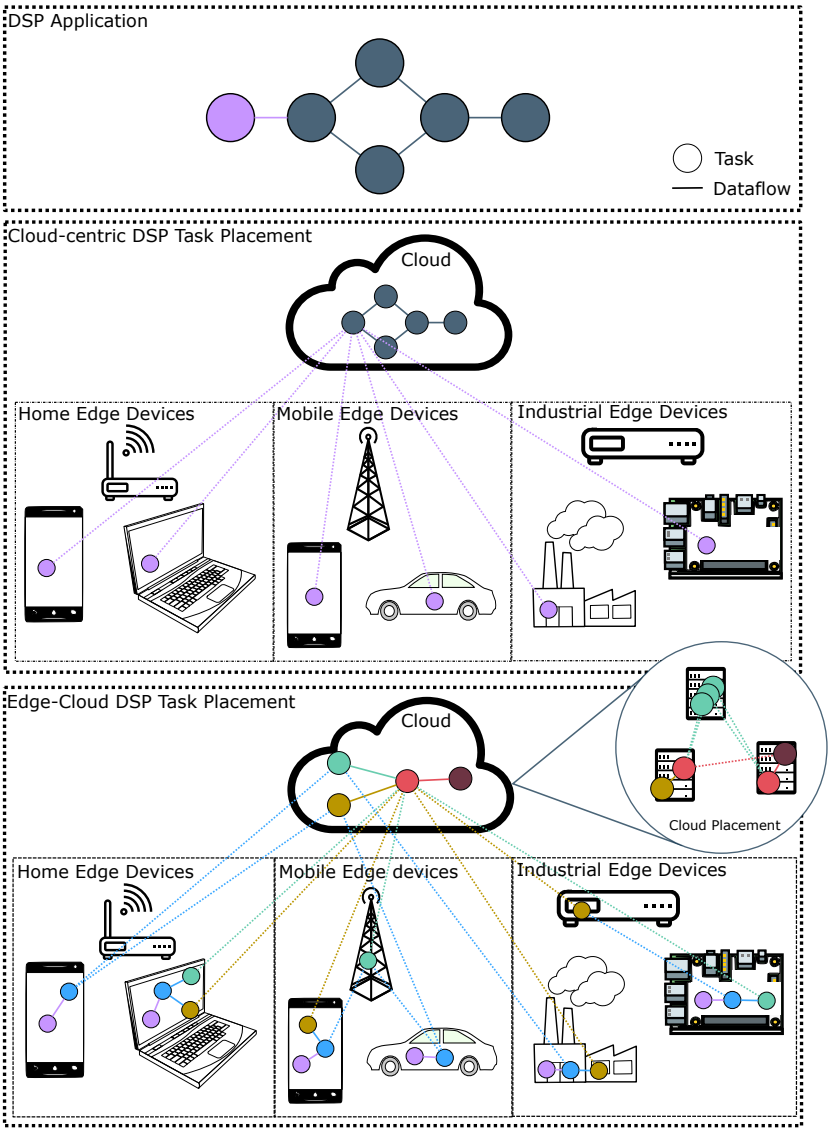


FIGURE 2.1: Overview of Cloud-centric and Edge-Cloud Task Placement for Distributed Data Stream Processing Applications. Each dotted line egressed from a task indicates the dataflow over the logical connectivity established via Local Area Networks and Wide Area Networks. A purple coloured task represents an event generation source where the generated events are then processed at the other tasks down stream.

- Classify and discuss the state-of-the-art techniques, algorithms, methodologies and the platforms for data stream processing on edge-cloud resources based on our taxonomy.

2.1 Background

Data stream processing is a versatile methodology for processing continuous data streams. Deploying DSP platforms on emerging edge computing resources and well-established

Cloud Computing resources has become a viable approach to overcome various shortcomings of the traditional data stream processing. In this section, we discuss the evolution of the edge and cloud computing paradigms along with the emergence of data stream processing in order to position our literature survey on Distributed Data Stream Processing on edge and cloud computing environments. We initiate our discussion based on the timeline depicted in Figure 2.2.

2.1.1 Evolution of Edge Computing and Related Infrastructure

Cluster Computing emerged to improve the parallel processing of data by connecting multiple tightly coupled computers over a Local Area Network to work as a single computer after Amdahl [22] presented the case for parallel processing in 1967. A cluster appeared as a single integrated entity that provided computing resources. During early 1990s, *Grid Computing* evolved from cluster computing which enabled sharing and aggregation of geographically distributed, loosely coupled, computing resources or clusters automatically and dynamically at runtime. *Cloud Computing* originated from these concepts of grid computing in 2000s with the promise of providing computing resources as a utility over the Internet [23]. In 2006 Amazon released the first commercial cloud service platform that included *Elastic Compute Cloud (EC2)* and *Simple Storage Service (S3)*. Cloud is a collection of virtualised, inter-connected computing resources that are presented on demand based on a pay-as-you-go cost model, with the help of Service Level Agreements (SLAs) to govern the quality of the service provided. With the emergence of cloud computing, the computing paradigm has completely changed where cloud has become the first choice for many applications ranging from education, healthcare, manufacturing, mobile technology, to research. Main advantages of cloud computing are the scalability, availability, reliability, versatility, and the long term economical value whereas disadvantages being security and privacy concerns, and high latency caused by the communication overhead. Today, the worldwide public cloud computing market remains a 250 billion dollar industry [24].

Meanwhile, in 1999 the idea of Internet of Things (IoT) was proposed for supply chain management [25] with the concept of utilising ubiquitous, computation capable, devices for conducting local computation. The concept of IoT has now moved to making any ubiquitous device computationally capable and connected to the internet, thus becoming *smart* and able to conduct local computations. However, due to the lack of processor, memory, battery and networking technologies at the time, IoT devices remained quiescent for nearly a decade. As these technologies evolved, mobile devices slowly became resource rich, and the sustained efforts to utilise mobile computing led to two options. One option led to web based services where mobile devices were only thin clients, and

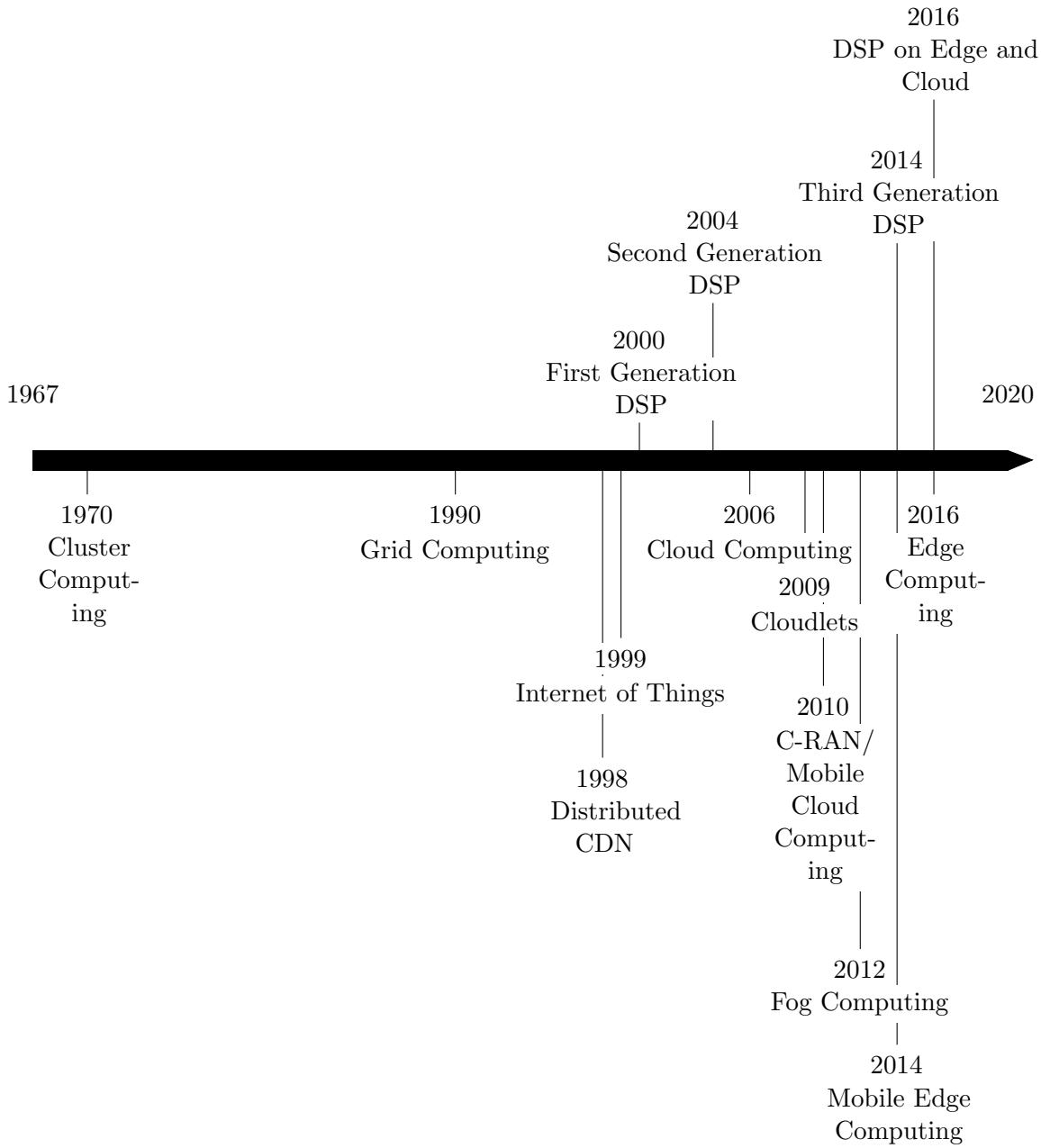


FIGURE 2.2: Timeline of the evolution of the data stream processing (top) and the evolution of cloud and edge computing paradigms (bottom).

the other option led to ubiquitous computing. As a result, in 2009, Satyanarayanan et al. [26] presented a composition of already established cloud computing and emerging mobile computing paradigms, referred to as *Cloudlets*. As the mobile devices during that era were not significantly resource-rich, in order to conduct real-time computations, cloudlets presented resource-rich computing resources that were one hop away from the mobile devices over a wireless LAN instead of relying on remote cloud computing resources. The cloudlet architecture introduced the idea of task offloading to nearby computing resources for multi-user mobile computing applications, and placed the idea for edge computing, which the authors themselves formally defined later in 2017 [27]. By 2010, as the mobile devices became more resource-rich, and with the growth of smart phone users, cloud computing needed to adapt to these changes by introducing *Cloud Radio Access Networks (C-RAN)* [28] [29]. C-RANs augmented the traditional cellular network architecture on the cloud which allowed the cloud infrastructure to support RAN functions. In a similar effort, Kumar and Lu [30] proposed *Mobile Cloud Computing* where tasks are offloaded to cloud from the mobile users to save time and energy.

As the capabilities of devices located outside the cloud grew increasingly, in 2012, Cisco coined the term *Fog Computing* as a response to overcome the shortcomings of cloud-centric processing paradigm by bringing some of the processing away from the cloud to the network, with the focus mainly on IoT services [31] [32]. The main advantages of Fog Computing are low latency due to proximity to the users, location awareness, mobility support, and geographical distribution. However, the proposal for Fog Computing remains focused on the scalability of the IoT infrastructure instead of the capabilities of the devices themselves [27] where hierarchical layers of fog nodes provide services between IoT edge devices and the cloud. In 2014, *Mobile Edge Computing (MEC)* was standardised by the European Telecommunications Standards Institute [33]. Conceptually similar to Fog Computing, MEC focused on improving the computation capabilities of the Radio Access Network infrastructure that are inherently in close proximity to the users and devices, as optimising for low latency and high bandwidth continued to be the objective.

As defined by Shi et al. [34] in 2016 and Satyanarayanan [27] in 2017, *Edge Computing* present a complementing computing paradigm to cloud computing by utilising the devices located outside the cloud, at the edge of the network and closer to the end users for processing as these devices have become increasingly capable. Edge computing can be defined as any computing or networking device between the data sources and the cloud data centres, that has the capability to conduct some form of processing on the data, thus enabling the processing of downstream data on behalf of cloud services and

upstream data on behalf of IoT services [34]. With the many similarities between Cloudlets, MCC, Fog Computing and MEC, we have observed that the community has not yet converged on a common term as evident in [35], [36], [37], [38], [39], [34], [32], [40], [41]. As a result, throughout this study edge computing will be used as an umbrella term to cover the broad definition for any computationally capable device logically located outside the cloud computing tier. Similar to Shi et al. and Satyanarayanan, our perspective is that edge computing and Fog Computing have a large conceptual overlap albeit with different focuses. Further, we consider MIST computing [42] to be another subset of the edge computing paradigm as they support analytics applications in the context of IoT.

We also believe that the commercial interest conveyed by the industry such as Google's implementation of artificial intelligence at the edge [6], Amazon's AWS IoT for the Edge and the AWS IoT Greengrass platform [7], Microsoft's Azure IoT Edge [8], in-device video analysis devices [9] [10] [11], home automation devices [12], and industrial automation applications [13], along with the vast interest shown by the researchers, present a strong case for materialising the edge computing paradigm.

2.1.2 Evolution of Data Stream Processing

Data Stream Processing is a programming paradigm that evolved from the continuous query processing platforms for database management systems of early 2000s. A DSP engine (DSPE) is designed for real-time processing of an unbounded stream of events, and is responsible for executing a DSP application on the resources the engine is deployed on and to manage the execution according to the policies set by the application owner. We analyse the evolution of DSP here as categorised by Assunção, Silva Veith and Buyya [43] and illustrated in the timeline of Figure 2.2.

The first generation DSP paradigm emerged from the database management systems in 2000s with the need for executing long running queries on batch data stored in databases. NiagaraCQ [44], STREAM [45] [46] and Aurora [47] are first generation DSPEs that added support for Continuous Query Language in order to allow the users to get updates from the database as new entries are added, without having to re-execute the same query. STREAM and Aurora arranged the query plans as Directed Acyclic Graphs (DAGs) where vertices were operators and edges represented how the operators were composed. STREAM also shows the earlier signs of task placement by incorporating a chain scheduling technique that placed the operators based on their memory usage and to adapt to the changes in data arrival rate. Aurora also supports a task placement mechanism referred to as train scheduling where tuples going to multiple operators are

batched together based on a non-linearity property in order to reduce the number of I/O operations conducted between processing.

Medusa [48] is a second generation stream processing engine that was introduced in 2004 and uses Aurora as the query processor with added support for load balancing, load shedding, and task placement based on the dataflow. Borealis [49] is another second generation stream processing engine that inherits query processing functionality from Aurora, that addresses multiple shortcomings of SPEs of previous generation. Borealis allows dynamic revising of query results so that entries that are added later or discarded due to missing the processing window, can be processed. It also allows dynamic query modification in order to adjust the queries in runtime depending on the load on the system or other requirements.

By 2014, the amount of events generated by various sources were becoming exponentially large and the widespread adaption of cloud computing presented a viable computing platform to process these events. As a result, the third generation DSP paradigm was developed in order to improve the extendability of the earlier generations by adding support for user defined functions and designed to be deployed on cloud infrastructure. As a result DSPEs such as Apache Storm [50], Twitter Heron (now Apache Heron) [51], Apache Flink [52], Apache Beam [53], and Apache Samza [54] were implemented. These engines have wide adaptability in real world applications and are being actively developed. However, with the emergence of edge computing and Internet of Things, the demand on the velocity, variety, and volume of events to be processed has increased as well as the demand for the applications that process these events in real-time. As a result, the fourth generation of highly distributed DSPEs have been introduced which can be deployed on cloud and edge computing resources. In order to utilise the edge computing resources, which are underpowered when compared with cloud resources, these fourth generation DSPEs need to be lightweight as well. SpanEdge [55], EdgeWise [56], and Apache Edgent [57] are modern DSPEs designed to be deployed on edge devices.

2.2 Taxonomy

Figure 2.3 illustrates the taxonomy of the Distributed Data Stream Processing in edge and cloud computing environments. We have classified the state-of-the-art in the following aspects:

- **Application**: the characteristics of DSP applications such as the application architecture, various requirements of the application and the service delivery method employed by the application are categorised.

- **Infrastructure**: the two main components of the DSP infrastructure, computing resources and the processing platforms are discussed.
- **Task Placement**: the task placement approach is divided and analysed in four subcategories – modelling strategy, optimisation objectives, placement controller positioning, and dynamism of the placement controller.
- **Evaluation Tools**: the different tools available for evaluating the DSP systems, task placement and their characteristics are probed.

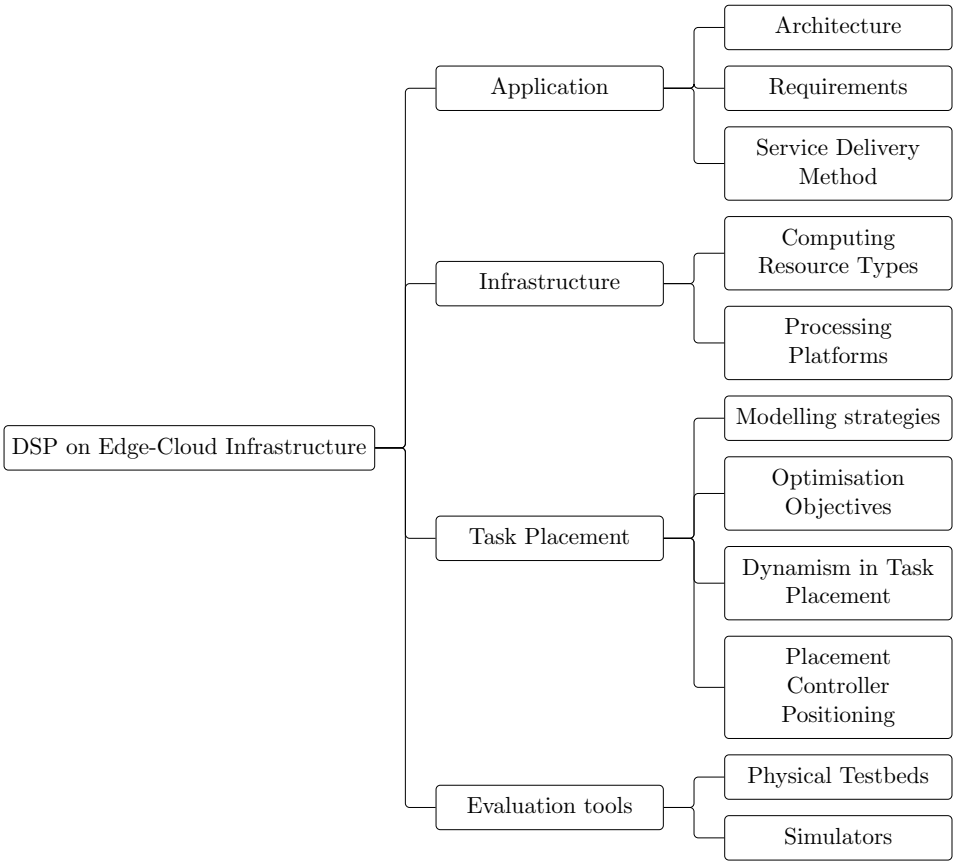


FIGURE 2.3: The high-level taxonomy of data stream processing and task placement on edge cloud infrastructure.

It is also important to note that we have only conceptually categorised the state-of-the-art here without the notion that these categories are mutually exclusive. For example, task placement is dependent on the resources provided by the infrastructure and as a result they need to be discussed together. Therefore, the surveyed papers in the following sections may belong to multiple subcategories and may be discussed and referred multiple times with respect to the relevant category in the taxonomy, for completeness.

Salaht, Desprez and Lebre [58] present a similar taxonomy where they classify the service placement problem for fog and edge computing into four main categories based on the

problem statement, service placement design, optimisation strategies, and evaluation environments. They have identified that the evaluation tools available in this domain need to be identified, similar to our classification. However, the focus of their categorisation is on the formulation of the service placement problem whereas we focus on a high-level perspective with applications, resource infrastructure, task placement problem, and the evaluation tools as categories.

2.3 Application

Assunção, Silva Veith and Buyya [43] describe a Data Stream as one or more types of input data that arrives in high volume and velocity, thus often stressing the communication and computing resources. DSP applications are being used in many different edge computing scenarios such as fitness tracking [59], federated learning [60] or virtual/augmented reality applications [61]. As described earlier, a typical distributed DSP application consists of sources, operators, sinks and dataflow between them. The application can be deployed onto one or more computing resources. Based on these characteristics of the applications and the solution provided to the end users, we have identified the following categories as shown in Figure 2.4.

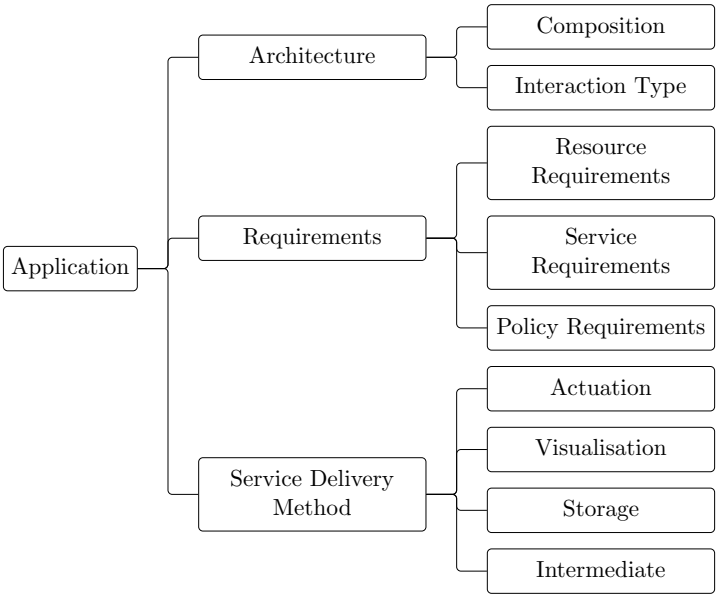


FIGURE 2.4: The taxonomy of data stream processing applications.

2.3.1 Architecture

We categorise the architecture of a DSP application into its composition and interaction type. Composition denotes how the components in the application appear to an

outside entity such as the stream processing engine. Interaction Type denotes how the application components interact with each other.

2.3.1.1 Composition

The composition of a DSP application or the functional layout of the application defines how much control the application developer, the user or the underlying application execution framework have in deployment of the application. The composition of a DSP application can be either *Monolithic* or *Distributed*. In order to understand the contrasting characteristics of these two categories, lets consider a video processing application that processes a stream of video frames to count the number of people entering a shopping mall which needs to conduct multiple tasks to acquire the desired end result such as (but not limited to) sampling the video feed, processing a frame, getting the bounding boxes per frame, extracting features from bounding box, tracking, counting, and visualising the results.

Monolithic DSP applications appear as a single component that are not designed to be logically separated. A monolithic video processing application appears as a single application that takes a video feed as an input and produces the people count visualisation as the output. Due to this inseparable nature, monolithic DSP applications become infeasible to be deployed on to multiple resources unless migrating the whole application is applicable by utilising replication or a time sharing approach. Tinini et al. [62] consider the BaseBand Units of Radio Access Networks as monolithic applications to be placed between cloud and fog networks to improve the energy efficiency and network capacity. Placement of multiple monolithic applications on to different computing resources is studied in [63] and [64] where Mahmud et al. [63] focus on improving the quality of service using fuzzy logic models where as Tan et al. [64] focus on improving the response time using an online approximation algorithm. Souza et al. [65] show the placement of a monolithic image processing application on to IoT and cloud resources where each application replicated on each resource is responsible for complete processing of a single image which shows how data parallelism can help with monolithic DSP applications.

Distributed DSP applications appear as a single application that can be logically separated into multiple individual components that perform a well defined task, arranged in a pipeline, and has well defined input stream(s) and output stream(s) which are then fed to the next task in the pipeline. If we consider a distributed video processing DSP application, the task that samples the video feed takes an input stream of video frames and outputs a sample of that stream where each frame is then fed to the frame processing task. As a result of this separability, the tasks themselves can be located on different

processing resources after carefully connecting the input and output data streams and such applications are widely considered in the literature as the most feasible type of applications that are supported by various DSP frameworks. In studies such as [66], [67], [68], [69], and [70] the authors consider distributed DSP applications as a collection of loosely coupled services that can be placed on IoT or cloud resources and composed to communicate with other services to complete the tasks.

Distributed DSP applications are also often organised as Directed Graphs (mostly acyclic) where the tasks represent the vertices, and the input and output connections represent edges. The main advantage of this approach is composing the modules in a directed graph allows the application deployment frameworks to globally optimise the application execution. Directed Acyclic Graphs (DAGs) are used for composing the distributed DSP applications in [18], [71], [72], [20], [73], [74], [75], [76], [77], [78], [79], [19] and [80]. Taneja and Davy [81], and Mahmud et al. [82] use applications of Sense-Process-Actuate model composed as directed cyclic graphs, where the IoT devices sense the surroundings, then send the stream of events to the processing tasks and the processed decision events are then sent back to the actuators to take an action.

Notably, Kim et al. [83] introduce a novel method to extract the edge executable components from a monolithic application so that the application can be distributed on to other resources by analysing and transforming the programme into sub-programmes during compile time with guaranteed correctness.

2.3.1.2 Interaction Type

DSP applications with multiple tasks that may be located on different computing resources require the tasks to interact with each other. The method of interaction depends on the application as well as the underlying infrastructure. We have identified that the existing literature utilises message passing techniques such as shared storage, web services or publish-subscribe model.

Shared storage is useful for applications that are deployed on a distributed infrastructure across multiple computing resources. Arkian, Diyanat and Pourkhalili [42] use a distributed database that stores both application data and meta-data to aid the interaction between services asynchronously.

RESTful and other web services are also widely used in proposed solutions to conduct the interactions between the multiple tasks deployed on different resource types due to the readily available language support, ease of management, and because of the concepts such as micro-services that have become increasingly popular to realise the Service Oriented Architecture. These services are being used as both synchronous and asynchronous

clients depending on the application. Mass, Chang and Srirama [84] propose a workflow management for edge devices that utilises HTTP based web services asynchronously to interact between components. Similarly in [85] and [86], the interaction between the services is modelled as micro-services communicating over a RESTful web service.

A common interaction method between the tasks in a DSP is to use the publish-subscribe model by utilising queues (one to one) or message brokers (one to many) between the components. This architecture increases the modularity by decoupling the relationships between the tasks thus allowing fine grained control of their placement. The brokers are either built into the underlying execution framework itself or configured and deployed by the application developers. Studies such as [87], [1], [88], [89], [72], [90], [91], and [92] illustrate the use of this model for communication between the tasks deployed on different edge, fog or cloud nodes. DSPEs such as Apache Storm [50] and Apache Flink [52] use queues as read/write buffers internally. Here, when two tasks on two separate devices need to send an event stream, the sender's events are put to a writer queue which are then serialized over to the network and read from the network on the receiver's side and put into the reader queue to be consumed by the task.

2.3.2 Requirements

A DSP application may have one or more resource requirements, service requirements or policy requirements depending on the use case. These requirements are defined by the application owners depending on the available resources. Optimisation objectives may also arise from requirements as further discussed in Section 2.5.2. Task placement modelling strategies discussed in Section 2.5.1 translates these requirements to the logics of making the task placement decision.

2.3.2.1 Resource Requirements

Resource requirements of a DSP application is concerned about the physical or virtual services available with the underlying infrastructure that consists of cloud and edge computing resources as well as the network(s) that connects the resources such as the CPU, memory, bandwidth, power consumption, GPU, storage, virtual machines, containers, sensors, or actuators. For example, in a video processing DSP application, the video capturing task may require high memory, CPU, GPU and bandwidth in contrast to a frame processing task which may require comparatively less resources.

Measuring the resource requirements of DSP tasks is a key component in task placement as the resources the tasks are placed on should be able to satisfy these requirements. As

a result, quantifiable requirements such as the CPU load is measured in Instructions per Second (IPS), memory and storage in Bytes, bandwidth in Bytes per second, and power consumption in Watts. Other non quantifiable requirements can be measured based on their availability.

Virtual machines and containers as resources have a cost involved with them depending on their location – such as in a managed cloud. Therefore, in such cases the cost and service level agreements of the cloud provider need to be factored during task placement [61].

Peng et al. [18], Santos et al. [67], Taneja and Davy [81], and Xia et al. [93] consider resource requirements such as CPU, memory, storage and bandwidth for making the placement decision. In [71], the authors consider the availability of GPU resources while making the task placement decision where as Cardellini et al. [73] partially consider the availability of nodes to make the placement decision. In addition, studies such as [94], [92], [95], [96], [97], [98], [99] and [100] consider the energy or power consumption requirements of the DSP applications as low power consumption is a critical requirement for many edge devices.

2.3.2.2 Service Requirements

The quality of the service required by the end user of a DSP application is measured by various metrics such as the latency, throughput, energy consumption, service availability, resource utilisation, or quality of experience. The main difference between the metrics for service requirements and the metrics in optimisation objectives is that the requirements only demand satisfaction of a metric against a threshold value, whereas optimisation objectives require either minimising or maximising the metric to achieve optimality. Therefore, all the objectives set for optimisation implicitly become service requirements. In addition to that, some studies have focused only on constraining the satisfaction of requirements without any optimisation as depicted in [101], [102], [103], [104], [105], [106], and [107].

2.3.2.3 Policy Requirements

In addition to the resource requirements, some DSP applications define various other non-functional requirements that need to be addressed. Privacy is a major requirement in many such applications, where some components of the application need to be placed in a particular device or a region in order to preserve the privacy of the data being processed. For example, a text prediction application based on the typing pattern of a

mobile user, shall require the text typed by the user processed on the device itself instead of a remote cloud. Shi et al. [34] discuss multiple applications with privacy policy requirements such as a crowd sourced video analytics application for finding a missing child where the video feed needs to be processed locally unless an approximate identification is found. The authors also illustrate other smart home, smart city, healthcare, insurance, and government applications that involve edge and cloud devices, and have privacy policy requirements. Similar privacy related policy requirements are discussed in [93], [105], and [108]. Some tasks in a DSP application needs to be collocated on to a single computing resource in order to acquire the optimum performance such as tasks that share a common source where on the other hand some tasks may need to be placed mutually exclusive from one another due to fault tolerance requirements as described in [109]. Imposing similar policy requirements can be done with less effort due to the distributed nature of DSP applications and the location awareness of the edge infrastructure. However, we have identified that this aspect is not sufficiently explored in the existing literature.

2.3.3 Service Delivery Method

The service delivery mechanism of a DSP application is how the application delivers the outcome to the end user or the application that requested it. We have classified the service delivery mechanism in four main categories to capture the DSP applications deployed on edge and cloud resources – actuation, visualisation, storage, and intermediate. Low latency becomes a key requirement in delivering services as actuations or visualisations. It is also important to note that one or more service delivery methods can be used in a single DSP application depending on the use case.

2.3.3.1 Actuation

Service is delivered as an actuation, when a DSP application processes a stream of events related to a surrounding received from a sensor or another source, and produces an end result which may be a decision to make a change on the surrounding or on another physical device such as moving a robotic arm, closing a valve, panning a camera or increasing the rotations per minute of a fan. Ravindra et al. [110] show a smart city application that uses various actuators such as water pump controllers and Pan-Tilt-Zoom cameras which act as the sinks of DSP applications. Rahmani et al. [111] present IoT based critical healthcare applications such as adjusting the Insulin pump rate by processing a stream of blood glucose level readings of a patient or electrical nerve simulation based on the heart rate, which are also notably reliant on millisecond

response times. Fitzgerald, Pióro and Tomaszewski [112] discuss another latency critical DSP application that could be deployed in a nuclear reactor management system which can open and close valves to flood the reactor with water by processing a series of temperature readings to protect the reactor from overheating. Similarly, the works in [97], [113], [83], [114], [1], and [115] consider DSP applications that deliver end results to actuators.

2.3.3.2 Visualisation

In visualisation, the end results are displayed to the end user via graphs, overlays, alerts or via other visual media which is a service delivery method frequently used in analytics, surveillance, and augmented/virtual reality applications. Rathore et al. [116] present a real time microclimate analysis application that visualises the results generated by processing multiple streams of events produced by various sensors deployed in an urban environment. Banos et al. [117] and Firouzi et al. [118] present frameworks for healthcare applications to visualise data acquired by processing information coming from health related sensors. Taneja and Davy [81] illustrate how a DSP based analytics application delivers the end results by visualising on a display. Toshniwal et al. [50] describe how the Apache Storm stream processing engine uses stream processing techniques to generate visualisations for topology metrics. Further, the works in [119], [120], [93], [1], [121], [90], [31], and [122] show the use of visualisation as a service delivery method in a multitude of DSP applications operating on edge and cloud computing environments.

2.3.3.3 Storage

When the service delivery mechanism is storage, the end results are stored on a file system or a database which may be shared between multiple resources and may be located locally or remotely. The DSP applications that store processed results often conduct tasks such as aggregation or summarising and are less reliant on the low response times. Concone, Lo Re and Morana [120] utilise a human activity recognition application based on video processing and stores the processed data in remote storage. Rahmani et al. [111] discuss the use of local as well as cloud storage for storing processed results of healthcare applications for further analysis. AbdElhalim, Obayya and Kishk [97] use a DSP application that reads a stream of values from IoT devices and stores the processed results in the cloud. In T-Storm [123], the authors use a word-count application and a log stream processing application that delivers the final processed values to a MongoDB database storage. Storage based delivery mechanisms for DSP applications are also discussed in [1], [18], [124], [95], [58], [125], [92], [93], [126], and [117].

2.3.3.4 Intermediate

Intermediate service delivery mechanisms are concerned about the DSP applications that act as intermediaries between other applications, where the resulting stream is often fed to another application. The most common type of intermediate delivery mechanism is Extract-Transform-Load (ETL) applications that pre-process a stream of data and then output the resulting stream to another application either via a message broker, a web service or a shared storage as shown in [1], [110], [127] and [128]. In another example, Cardellini et al. [73] illustrate a top route ranking application that reads a stream of New York Taxi routes and produces a globally ranked top 10 routes of last 30 minutes and publish the results to a RabbitMQ message queue, which can be used by another DSP application as its source. Intermediate service delivery applications such as ETL applications can become ideal to be deployed at the edge for collecting data from multiple sensors, extract and upload only the filtered data to the cloud in order to reduce the bandwidth usage. The downside of this approach is the lack of complete records maintenance, as only a selected few events would be sent to the cloud, which can be mitigated by other means such as local logs and aggregation.

2.4 Infrastructure

Infrastructure provides the support that is needed to execute an application on available computing resources. We have identified that it can be categorised into two main categories such that Resource Types denote the types of physical or virtual computing resources available where as the Processing Platforms denote the solutions deployed on the resources to support and manage the execution of a DSP application. As illustrated in Figure 2.5, each category is further divided into subcategories depending on the key characteristics.

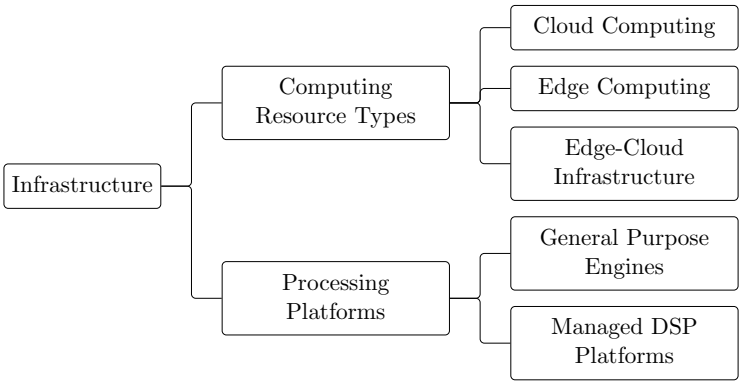


FIGURE 2.5: The taxonomy of the infrastructure for data stream processing.

2.4.1 Computing Resource Types

Computing resources provide the basic functionality to execute an application such as processing cycles, memory, storage, energy, and connectivity to other resources. Cloud computing emerged as a viable computing paradigm with ample resources for virtually any computing task with the need for adequate connectivity to manage the resources remotely. As the processing technologies evolved, devices logically located outside the cloud began gaining substantial processing power while being under-utilised. As a result alternative computing paradigms such as edge computing, fog computing and mist computing emerged that brought the computing capabilities to ubiquitous devices. For executing DSP applications, all of these computing paradigms provide reasonable justifications depending on the use cases. We study the literature surrounding these computing paradigms with the focus on DSP and discuss why we have decided cloud and edge computing for most viable computing resource types for the latency critical DSP applications considered in our work.

2.4.1.1 Cloud Computing

Cloud computing provides on demand, pay-as-you-go computing resources for any entity that requires them [129] for a comprehensive set of tasks which are provisioned using virtual machines or containers. The physical cloud resources are often located in remote data-centres which are managed by cloud service providers. One of the key features of cloud computing is the resource elasticity, which allows organisations or users to increase or decrease the resource capacity vertically as well as horizontally depending on the computing requirements of the deployed applications. Elasticity helps to ensure that the expected quality of service is provided to the end users. Cloud services have a monetary cost attached to them where you can purchase resources based on the usage or reservation. The drawbacks of cloud computing are communication overhead, lack of location awareness, cost, and concerns for data privacy. Deploying DSP applications and DSPEs onto cloud computing instances has been the accepted practice as shown in [53], [18], [50], [73], [130], [52], [43], and [131]. While the amount of resources available at the cloud is virtually unlimited, the drawbacks of the cloud has affected the quality of service provided by the latency critical DSP applications that serve the unprecedented scale of data generated by the devices located outside the cloud [34] [39]. As a result, extensions to the cloud computing paradigm have been suggested by the state-of-the-art.

2.4.1.2 Edge Computing

Edge computing presents a complementing computing paradigm to cloud computing by utilising for processing, the devices located outside the cloud – at the edge of the network. Due to the advances in the processing and networking technologies, these devices have become increasingly capable all the while being located in close proximity to the data sources and users. As a result, edge computing presents a viable platform for processing data as also described in [35], [36], [37], [38], [39], [34], [32], [40], and [41]. The main advantages of edge computing are the proximity to the user, low latency caused by proximity, location awareness, mobility support for mobile applications, and scalability. The disadvantages are relatively low resource capacity such as processing power, memory, storage and battery power, network connectivity issues caused by mainly wireless media and the heterogeneity of the devices which can cause compatibility issues across different devices. There exist many latency critical DSP applications as mentioned earlier that can benefit from close proximity computing resources. However, utilising the edge devices exclusively is not always viable due to the aforementioned shortcomings. As a result, a trade-off is presented with hybrid edge-cloud networks.

2.4.1.3 Edge-Cloud Infrastructure

Hybrid edge-cloud infrastructure present a solution that overcomes the disadvantages of both platforms while gaining from the advantages. Executing a DSP application on an edge-cloud infrastructure is illustrated in Figure 2.1 where some of the processing is conducted at the edge devices with the help of computing enabled devices and the remaining tasks are offloaded to either nearby devices or the cloud. Edge devices such as Raspberry Pis along with public cloud instances have been used for running DSP applications in [1], [19], [80], [132], [85], [133], [88], [99], [134], [135] and [136]. Pahl et al. [137] present a Raspberry Pi based edge-cloud PaaS architecture. Mortazavi et al. [138] introduce a novel approach to hybrid edge-cloud architecture, termed as *Path computing* where they propose to utilise the computing and storage resources located between the path from the user device to the cloud data centres in a hierarchical manner. Ability to use a device that either generated the events or located closer to the event source to conduct partial processing can largely reduce the communication overhead between the edge and cloud while also preserving privacy of the data processed if applicable. However, the concerns for reliability, and the lack of a unifying framework can be identified as main drawbacks of edge-cloud networks which need to be further explored.

2.4.2 Processing Platforms

A DSP engine (DSPE) is designed for real-time processing of an unbounded stream of events, and is responsible for executing a DSP application on the resources the engine is deployed on and to manage the execution according to the policies set by the application owner. As categorised by Assunção, Silva Veith and Buyya [43], earlier stream processing engines emerged from the database management systems with the need to execute long running queries on batch data stored in databases. The next generation of stream processing engines introduced distributed deployment of the engines in order to utilise the resources. However, these engines did not have the ability to execute user defined custom functions. As a result, the third and fourth generation of DSPEs have been introduced which have the ability to execute one or more DSP applications with user defined functions on distributed resource infrastructure.

We have identified that these frameworks can be categorised into two main categories. *General purpose platforms* allow a user to execute virtually any DSP application that implements the application programming interface (API) exposed by the platform and allow the platform to be deployed on any resource infrastructure preferred by the user. *Managed solutions* expose DSPE as a service such that a user can deploy their own solutions on an established framework.

2.4.2.1 General Purpose Platforms

Apache Storm [50], Twitter Heron (now Apache Heron) [51], Apache Flink [52], and Apache Samza [54] are few current generation general purpuse DSPEs that follow the dataflow model where an application topology is arranged as a DAG of operators as vertices and dataflow as the edges. Apache Spark Streaming [139] is another DSPE that is built using the Resilient Distributed Datasets that enable running computations in memory. These engines are designed to be deployed on cloud infrastructure and as a result they are not designed to be lightweight. Lee et al. [140] has been able to deploy Apache Storm, Apache Flink and Apache Spark Streaming on an edge cluster of Raspberry Pi 3 devices. Apache Edgent [57], SpanEdge [55], EdgeWise [56], and Apache NiFi [141] are lightweight general purpose DSPEs, that are designed to be executed on resource constrained environments such as edge devices.

2.4.2.2 Managed DSP Platforms

Google Cloud Dataflow [142] provides a managed stream processing solution that implements the dataflow programming model and unifies streaming and batch processing.

Amazon Web Services Kinesis and Azure Stream Analytics are similar managed streaming data services that are focused on the cloud computing paradigm. These platforms allow application owners to deploy DSP applications and define policies on how to utilise the computing resources as needed. However, these platforms are not focused on utilising the edge computing resources. With the emergence of edge computing resources, we can observe a shift in the direction of managed stream processing solutions as AWS IoT Greengrass [7], Azure IoT Edge [8], IBM Watson IoT [143] and Google Edge TPU [6] platforms are introduced. These platforms focus on moving cloud based stream processing applications to edge devices that are also managed by the platform. For example, Azure IoT Edge contains three components: IoT Edge modules execute applications on containers, IoT Edge runtime manages the edge device, and the IoT Edge cloud interface allows edge devices to be managed from cloud [144]. Through the cloud interface a user can create an application and deploy it onto the edge devices. Similar functionalities are available with other managed edge based DSP platforms as well.

2.5 Task Placement

When a distributed DSP application needs to be executed on different computing resources such as Edge and Cloud devices, deciding the optimum strategy for placing the tasks of the application on the available resources to maximise the desired outcome becomes pivotal. As a result the main focus on existing literature is on utilising novel approaches to solve the task placement problem. However, optimal DSP task placement problem has been proven to be NP-Hard [20] [73] by proving that a simple form of the task placement problem can be reduced to the Partition problem [145] which is known to be NP-Hard. Peng et al. [18] also prove the NP-Hardness of the task placement problem by reducing the problem to a binary Knapsack problem which is NP-Hard. Even though the task placement and optimisation problem is known to be NP-Hard, it is possible to find polynomial time approximation schemes to provide sufficient results to achieve desired outcomes [18][146].

Generally, the workflow of the task placement problem is formulated by first selecting the modelling strategy, then deciding on the optimisation objectives based on the application requirements, followed by deciding how to handle the dynamism of the overall system, next deciding where to conduct the task placement task, and finally solving the task placement problem to achieve the task placement plans. These steps are categorised and explored as shown in Figure 2.6.

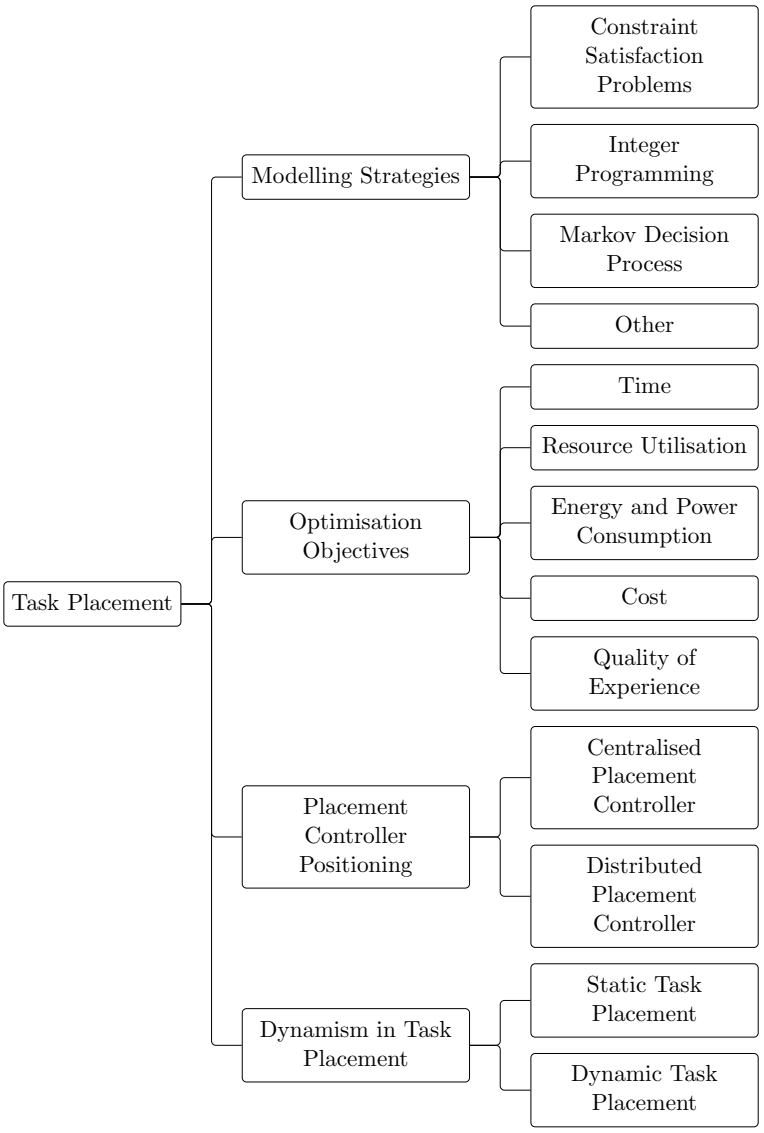


FIGURE 2.6: The taxonomy of formulating and solving the data stream processing task placement problem.

2.5.1 Modelling Strategies

The first step in designing task placement problems is deciding the modelling strategy. Commonly used modelling strategies include *constraint satisfaction and optimisation*, *integer programming*, *stochastic modelling methods*, such as *Markov decision process* or *stochastic network optimisation*, and various other strategies such as *minority games* or *assignment problems*. In the following sections we discuss the state of the art modelling strategies and compare the characteristics of these strategies.

2.5.1.1 Constraint Satisfaction Problems

A *constraint satisfaction problem (CSP)* provides a logical inferencing based methodology to solve task placement problem by first defining a set of variables, the corresponding set of domains and a set of constraints to be satisfied while mapping from variables to the domains. CSP has become a widely used methodology for modelling task placement problems due to the natural extendability of the approach as it is possible to add more constraints to the problem to fine tune the task placement solution provided. Then an objective function can be introduced to the problem, where by maximising the objective function, an optimal solution can be acquired for the CSP. Tools such as MiniZinc [147][148] and Choco Solver [149] are used as well as custom solutions for solving the CSPs. One of the main drawbacks of CSP is the need of having bounds for the domains of each variable in order to solve it, which results in additional work in some cases where such bounds need to be calculated when the environment conditions change.

Peng et al. [18] model the task placement problem as a CSP with constraints on the CPU, memory and bandwidth usage while optimising the solution for maximum throughput using a heuristic algorithm based on the Euclidean distance between the capacity and requirements in a 3D space. Thoma, Labrinidis and Lee [107] present a comprehensive set of constraints ranging from resource capacity constraints, collocation constraints, to policy constraints. The authors also provide empirical evidence that when compared with linear programming, satisfiability approach or brute force, CSP provides lower runtime in task placement problems, thus making it feasible for latency critical applications. Similarly, the works in [58], [81], [105], [106], [76], [104] and [126] utilise CSP approach for modelling the task placement problem with varying constraints and optimisation objectives.

2.5.1.2 Integer Programming

Similar to constraint satisfaction problems, *integer programming* is also a widely used methodology for mathematical optimisation where some or all of the variables are restricted to be integers. While *Integer Linear Programming (ILP)* methodology, where a linear optimisation objective function is used subject to a set of linear constraints, has been most common, non-linear approaches are also being used in the existing studies. IBM ILOG CPLEX ILP solver [150] is the most commonly used solver that is present in the literature apart from ad-hoc solutions.

Cardellini et al. [21] model the operator placement problem by considering the availability of nodes, cost of operator execution on nodes, response time and network QoS metrics

(inter-node traffic). They formulate an ILP to model the problem and optimize solving for both the replication and placement of operators in a DSP application. Skarlat et al. [69] use ILP to model CPU, RAM and storage as well as residency constraints while optimising for resource utilisation and QoS metrics such as task deadlines. Santos et al. [67] introduce a task placement based on ILP for a smart city management application by optimising for service availability, bandwidth usage, resource utilisation and energy consumption. Likewise [62], [151], [77] and [152] use ILP to model and optimise task placement problems for DSP applications.

An *Integer Non-Linear Programming (INLP)* problem consists of non linear constraints or an objective function in contrast to ILP. In FOGPLAN [153], the authors present a processing, storage, communication, service deployment, and service delay violations based non-linear optimisation to formulate an INLP which is then solved using two greedy algorithms to acquire placement plans to either minimise service violations or cost.

Some studies use *Mixed Integer Linear Programming (MILP)* to model the task placement problem where not all of the variables involved are represented as integers such that some of the variables can take real non-integer values. Ascigil et al. [154] propose a MILP based latency-critical task placement algorithm for edge cloud systems by optimising the placement for minimising the number of failed requests with constraints for flow conservation, resource capacity and response time. Zeng, Gu and Yao [70] utilise a MILP to model the task placement problem for energy efficient Cyber Physical Systems with source rate control, load balancing and replication with energy optimisation. However, to reduce the computation complexity they then relax the model to an ILP which is then solved in polynomial time using a heuristic algorithm. *Mixed Integer Non-Linear Programming (MINLP)* problems consider variables or optimisation functions that may be non-linear or can take non-integer values. To deal with the intractability of MINLP problems due to high computational complexity these problems are often solved by relaxing the problem into a MILP by linearising as depicted in [103], [155], [156] and [42]. However, Zhang et al. [157] propose an iterative search algorithm with a penalty function to optimise for computing overhead, channel allocation and power consumption in a MEC network even though the optimality is not guaranteed for solutions.

Notably, constraint programming and integer programming techniques are both utilised in existing literature to model the task placement problems by first mathematically defining the problem as variables, constraints and the objective functions. Constraint programming model works using logical inferencing where as integer programming relies on algebraic computation. Selecting which technique to utilise depends on the problem such as the type of constraints needed and the available tools.

2.5.1.3 Markov Decision Process

Stochastic modelling has proven to be another useful strategy for solving the task placement problem as it allows to model the system using probability and random variables which leads to realistic representation of the inherent characteristics. While *Markov Decision Process (MDP)* has been the prominent approach found in the literature due to multitude of reasons, other stochastic modelling approaches such as *stochastic network optimisation* [158] and *queueing theory* [73] are also present in the state-of-the-art.

MDP is effective in modelling the task placement problem as it lets the transition between the tasks to be designed as states of the model. Model-based MDP relies on identifying the transition probabilities as well as the rewards associated with transitions whereas a model-free approach assumes these are unknown and relies on *learning* the values. Due to this versatility of the model, if the application and system characteristics are well defined, a model-based MDP can be formed which can be then solved using dynamic programming. If the characteristics are not known beforehand, the problem can be modelled as a model-free MDP and *reinforcement learning (RL)* can be used to solve the problem. One of the main drawbacks of MDP is the state explosion where the problem becomes intractable with increasing complexity. However, different approaches are used in the literature to overcome this problem. Hou et al. [159] introduce a resource allocation algorithm for mobile video applications running on a vehicular cloud based on a social graph of connections between these vehicles by modelling the problem as a continuous time MDP (CTMDP) with a discounted reward function. Wang et al. [160] formulate a MDP framework for computation offloading for mobile edge computing. The MDP is modelled as a distance based problem where the distance between the user and the service location is used as the state to avoid the state explosion problem. Terefe et al. [161] use a static analysis and dynamic profiling of an application to model the energy consumption. Then they use MDP to develop an energy efficient multisite data offloading policy to offload data or computation intensive tasks of an application. Liu, Khoukhi and Hafid [162] focus on mobile data offloading for mobile cloud computing (MCC) architecture where data offloading happens over cellular, WiFi and device-to-device communication networks. The authors model the data offloading problem as a finite horizon MDP with a fixed reward and solve the problem using a hybrid offloading algorithm with two phases. In the first phase value iteration algorithm is used to solve the MDP and to acquire an optimal policy and in the second phase the optimal action is selected from the acquired policy based on the the current system state. Follow-Me Cloud (FMC) [163] enables service mobility over federated data centres based on an optimality criteria such as the geographical proximity or the load by utilising MDP with a model that trades off between the cost and quality. Similarly, the works in [94], [164],

[165], and [166] also present studies that utilise a model based MDP to solve the task placement problem.

In contrast to the model-based MDP, Lei et al. [167] propose a computation offloading algorithm for multiuser mobile edge computing (MEC) system deployed on edge computing resources that uses Narrowband IoT cellular technology for communication where they use the model-free linear value function approximation and temporal difference learning on the problem formed as an infinite horizon continuous time MDP with a weighted average reward function. Russo, Cardellini and Presti [168] model the problem of acquiring optimal elasticity policies as an infinite horizon MDP. They first use linear function approximation and tile coding to reduce the search space and state explosion and then learn the optimal policy using RL. Gazori, Rahbari and Nickray [96] model the task scheduling problem as a MDP to reduce the average End-to-End (E2E) service delay and computation cost under the resource and deadline constraints for fog devices, and use *double deep Q-Learning algorithms* for acquiring solutions. Further, [169], [170], and [171] present RL based solutions for a model-free MDP based task placement problems.

MDP presents a toolset for solving the task placement problem using multiple approaches. However, the state explosion which leads to the solution becoming intractable, needs to be mitigated in order to develop a scalable solution. The existing literature has not studied this approach sufficiently based on the characteristics of DSP applications running on edge-cloud infrastructure.

2.5.1.4 Other

Other strategies used for task placement include *game theory* based solutions such as in [97] where the authors use a minority game model to optimise the quality of experience delivered to users in a task offloading problem and in [172] authors design a task placement algorithm that can achieve a Nash equilibrium for multi-user mobile edge-cloud computing environments. Ni, Zhang and Yu [173] introduce another strategy by utilising *Priced Time Petri Nets* for task placement in a fog-cloud environment.

2.5.2 Optimisation Objectives

The optimisation objectives of a DSP application define the metrics that are considered by application developers and end users for measuring the quality of service delivered. For real-time DSP applications that process an unbounded events stream continuously, latency is the most critical optimisation objective. However, resource utilisation, energy

consumption, monetary cost, and quality of experience which is a subjective measurement is also considered in the literature.

2.5.2.1 Time

Time is the most critical Quality of Service optimisation objective for real-time DSP applications. Time is generally measured as latency by taking the time difference between the event egress time and either the event ingress time on the processing system or the event generation time at the source. If the incoming event can take multiple different paths on the topology, the time difference from the longest path is considered. Latency is composed of processing time at tasks, transmission time between the tasks and the waiting time in queues. However, time is measured in various forms in the literature in addition to end-to-end time difference, such as round trip time, number of hops, deadline awareness [75] or critical path length [83].

2.5.2.2 Resource Utilisation

Resource utilisation measures the amount of computing, or networking resources utilised by the application components during the execution. Resource utilisation also has an impact on the other metrics such as processing time where if a processor is over-utilised, the processing time increases due to context switches and throttling. While under-utilisation of resources does not have a critical impact on the application execution in a similar way to over-utilisation, it can negatively affect the efficiency and the monetary cost incurred for the un-used resources.

2.5.2.3 Energy/Power Consumption

Many edge devices are battery powered and as a result power consumption is a critical requirement. Optimising for power consumption is a primary objective in many studies such as in [103] and [174], and remains a secondary objective in some studies such as [170].

2.5.2.4 Cost

Monetary cost for operating edge and cloud systems is composed of processing costs, network costs, storage costs and other auxiliary operating costs. Leasing cloud computing resources is generally operated as a pay as you go model whereas edge computing

paradigm does not have an established platform as such yet. Therefore, in the state-of-the-art monetary cost for edge computing resources is estimated [155] [175]. In some cases, a monetary cost can be incurred regardless of the utilisation of the resource and as a result in such cases, optimising for resource utilisation is important in reducing costs. In addition, optimising for energy consumption can reduce the operating cost of devices as well.

2.5.2.5 Quality of Experience

In contrast to the Quality of Service metrics, which are driven via an agreement between the user and the application, Quality of Experience (QoE) is concerned about the user's requirements and perceptions which is a subjective measurement [63]. Various QoE related objective functions are considered in the state-of-the-art such as service coverage, response time and network congestion [63], reduction in latency and energy [176], service resolution [177], and communication and computation rating [178].

When only one of the optimisation objectives is considered for the task placement, the problem is referred to as a mono-objective optimisation and on the other hand multi-objective optimisations attempt to optimise more than one objective. We summarise the different optimisation objectives and classify them as either mono- or multi-objective optimisations in Table 2.1 and Table 2.2 respectively.

Apart from the key metrics noted here, few other optimisation objectives can also be observed in the literature such as availability in [78], [75] and [154], minimisation of service migrations in [67], and stability, i.e., the probability of completing a task in [182].

2.5.3 Dynamism in Task Placement

The dynamism of the task placement describes how the task placement algorithm handles the changes in the overall system, i.e., the underlying resource infrastructure or the deployed applications. Static task placement solutions rely on a fixed task placement plan that is generated using application and resource conditions acquired either by definition, pre-existing knowledge or prediction whereas dynamic task placement solutions adjust the task placement based on the ongoing changes of the overall system or a component of the system.

TABLE 2.1: Mono-objective optimisations presented by the state-of-the-art in DSP on edge and cloud environments.

Study	Objective(s)				
	Time/ Latency	Resource Utilisation	Energy Consumption	Cost	Quality of Experience
[179]	Sum of maximum computing latencies	–	–	–	–
[83]	Critical path latency	–	–	–	–
[88]	Response time	–	–	–	–
[19]	End-to-end latency	–	–	–	–
[180]	No. of latency violations	–	–	–	–
[124]	Average latency	–	–	–	–
[98]	Average latency	–	–	–	–
[81]	–	Resource utilisation (CPU, memory, bandwidth)	–	–	–
[69]	–	Fog utilisation	–	–	–
[152]	–	–	–	–	Number of satisfied requests
[106]	–	Fog utilisation	–	–	–
[112]	–	–	Total energy Per-node energy	–	–
[161]	–	–	Processing and transmission energy	–	–
[70]	–	–	Processing and transmission energy	–	–
[42]	–	–	–	Monetary cost	–
[181]	–	–	–	Monetary cost	–
[63]	–	–	–	–	Rating of Expectation
[178]	–	–	–	–	QoE (Computation and Communication)
[177]	–	–	–	–	Service resolution

TABLE 2.2: Multi-objective optimisations presented by the state-of-the-art in DSP on edge and cloud environments.

Study	Objective(s)					
	Time/ Latency	Resource Utilisation	Energy Consumption	Cost	Quality of Experience	Other
[182]	Service latency	–	–	–	–	Stability
[80]	Sum of the makespans, Schedule planning time, Total stabilization time	–	–	–	–	Total no. of migrations
[183]	Avg. response time	–	Energy loss	–	–	–
[100]	Response time	–	Energy efficiency	–	–	–
[184]	Service latency	Resource utilisation	–	–	–	–
[185]	–	Fog utilisation	–	Monetary cost	–	–
[67]	Hop count, Path loss	Bandwidth, No. of active comp. nodes	Cloud	–	No. of IoT app. requests	No. of service migrations
[167]	Average latency	–	IoT	–	–	–
[75]	App. deadline satisfaction ratio	–	–	–	–	Availability
[186]	Total latency (Comp. and comm.)	–	–	Migration cost	–	–
[90]	End-to-end latency	Bandwidth	–	–	–	–
[103]	Comm. latency	–	Fog and cloud	–	–	–
[68]	App. deadline	Fog utilisation	–	–	–	–
[187]	App. latency	Resource usage	–	Monetary cost	–	Throughput
[21]	Response time	Network	–	Monetary cost	–	Availability
[174]	App and queueing delay	–	Fog	Monetary cost	–	–
[188]	Latency	–	–	Monetary cost	–	No. of users
[176]	Aggregate latency	–	IoT	–	QoE (energy and latency reduction)	–
[189]	Network latency	Bandwidth, VM load	–	–	Quality score	–
[170]	Latency	–	IoT, Fog and Cloud	–	–	–
[190]	Transition time	–	–	–	–	Transition overhead

2.5.3.1 Static Task Placement

Static task placement is used based on the presumption that the status of the underlying infrastructure and the characteristics of the application remain constant during the execution of the application. If such a condition exists, then the optimised or satisfying task placement plan needs not to be changed. One of the main advantages of this approach arises in real time applications that require milli-second latency in generating results. If the task placement plan is to change, then the application execution needs to be paused, the tasks migrated to resource nodes based on the new plan and the application resumed. However, since DSP applications that are being considered here process an unbounded stream of events, pausing the application significantly affects the overall latency of generated results, thus pausing becomes infeasible for real-time applications. Naively, the disadvantage of static task placement arises when the overall conditions change where the current task placement plan may not be optimal or may not even satisfy the requirements. Static task placement strategies are used in [126], [191], [152], [79], [76], [192], [193], [93], and [81].

2.5.3.2 Dynamic Task Placement

Dynamic task placement requires constant feedback from the infrastructure and the application and based on this feedback the optimal or satisfying task placement plan is generated. The frequency of updating the placement plan impacts the overall quality of service delivered to the end user. As mentioned earlier, changing the task placement has an impact on the end-to-end latency notably for real-time DSP applications, and on the other hand optimal task placement can improve the QoS metrics. Therefore, a trade-off needs to be established between the frequency of task placement deployments and the gain in application objectives. Ghosh, Komma and Simmhan [80] present a heuristic based dynamic task placement with respect to the changes in the number of active applications running on the system. The authors also consider optimising for task placement planning time and waiting time for the events during task placement migration. Similarly, TCEP [190] present a dynamic task placement model that minimises transition time and transition overhead while satisfying varying QoS demands on latency and message overhead of the application. Follow Me at the Edge [186] is a mobility aware task placement solution that is designed to consider the performance-cost trade-off by optimising for end-to-end latency with a constraint on the long term cost budget. Wang et al. [160] present a dynamic task placement technique based on a Markov decision process for mobile users in a MEC environment. MigCEP [74] present a live task migration based on the mobility of the users and the devices using a time graph data structure. Saurez et al. [66] introduce two task placement algorithms: a proactive

task placement based on a self monitoring mechanism where the node itself initiate a task placement migration if the load is above or below a threshold and a reactive task placement based on the perceived load on other device nodes by neighbouring nodes. In [78], the authors have created a dynamic placement controller that monitors the applications where depending on the requirements, the scale in or out decisions are made. Likewise, [188], [194], [195], [177], [151], [17], [196], [67], [197], [198], [199], [65], [200], [154], and [201] introduce dynamic task placement approaches that adapts to the changing environment and application conditions. We have found that based on these prior work, a fully dynamic task placement approach where optimal task placement decision is made based on the current conditions, followed by the task placement migration, to be infeasible for real time DSP applications. Therefore, alternative methodologies to compromise the latency optimisation and dynamism needs to be found.

2.5.4 Placement Controller Positioning

Placement controller handles where the task placement plan is executed on the computing infrastructure, i.e., the host network. A *centralised placement controller* has either a dedicated computing resource, in the edge or cloud layer which has access to all the available devices and can control application deployment. A *distributed placement controller* hands over the execution of the task placement to devices themselves, thus removing the need for a centralised controller. It is important to note that depending on the positioning of the placement controller in the overall system, the surrounding framework for facilitating the operations of the control planner, such as monitoring the device capacity or monitoring the ongoing status of the tasks, also needs to be setup [15].

2.5.4.1 Centralised Placement Controller

Centralised placement controllers execute the placement plan from a device that can control the application deployment on all the available resources. The main advantage of having a centralised placement controller is the potential to achieve a globally optimal placement since the controller can observe a global view of the deployment. However, having a centralised placement controller requires the devices to report to the placement controller periodically which adds a communication overhead which may not be ideal for low powered edge devices and as the number of devices grow, the computational complexity grows as well. Further, to avoid having a single point of failure, generally one or more redundant placement controllers need to be placed in active-active or active-passive mode as depicted in [202]. In [83], [120], [81], [193], [119], [80], [188], [190],

and [203], the central placement controller is located in the cloud layer in an edge-cloud environment. [86] and [152] present a centralised placement controller that can be placed on any computing node based on a Kubernetes cluster. Donassolo et al. [78] present a centralised placement controller called FITOR orchestrator, that can be deployed on any fog or IoT device which works by extracting metrics about the nodes and the connections, and then greedily placing the tasks on to nodes. LAVEA [88] is an edge computing platform which contains a centralised placement controller known as offloading service which runs on the master *edge-front* node which is connected to the same network as the other edge clients, and manages the task placement onto other edge and cloud nodes. As mentioned earlier, the Apache Storm DSP engine [204] contains a centralised placement controller node known as *Nimbus* which also allows extending the placement controller to support versatile placement strategies as shown in [18], and [205]. Similarly, other processing engines such as Apache Flink consist of a node referred as *Job Manager* with the responsibilities of a centralised placement controller which can be extended for other placement controller implementations as prototyped in [206]. Both Nimbus and Job Manager controller nodes can be replicated for high availability.

2.5.4.2 Distributed Placement Controller

Distributed placement controllers are deployed on more than one device and has authority of controlling limited to a local scope in contrast to the global scope in a centralised controller. As a result, distributed placement controllers often result in locally optimised solutions. Advantages of distributed placement controllers include the tolerance of device failures, scalability, and low communication overhead and fast resolution of placement plans due to the reduced scope. Mahmud et al. [82] present a decentralised task placement controller that meets the latency as well as resource optimisation requirements by deploying a controller on every node. The controller maintains a list of tasks deployed on itself and another list of tasks deployed on other nodes in order to maintain the event routing according to the application topology. Fully distributed per device placement controllers are introduced by Singh et al. [105] where the authors present RT-SANE algorithm which has privacy awareness as well as performance awareness. Yousefpour et al. [153] have developed a fully distributed placement controller that is located on each computing node where based on the waiting time to process a request, the node itself forwards the request to a neighbouring node or to the cloud. Ottenwälder et al. [74] present a constraint on when the distributed placement controller would result improved network bandwidth usage depending on the frequency of the updates and the size of the placement plans. In addition, studies by Skarlat et al. [207], Venticinque and

Amato [106], and Minh et al. [68] present a hierarchically distributed placement controller where for a set of nodes there exists a common placement controller and multiple such sets of nodes congregate to build the computing infrastructure. Similar approach is presented by Shurman and Aljarah [191] where an edge based centralised placement controller is placed on each edge site, that traverses the available edge nodes in the local site logically organised in a tree structure according to the device capabilities and assign tasks depending on the requirements. This hierarchical approach has the advantage of being able to decentralise task placement while reducing the network load. In [20] and [73], Cardellini et al. have shown that a distributed task placement controller can be implemented in Apache Storm by deploying a distributed data store on each worker node to save their internal state and coordinate task migration. Tiwari et al. [208] present Merlin, a stream processing framework designed for hierarchical edge-cloud networks where they implement a decentralised placement controller using a store and forward mechanism. Here, the authors use a data store to store the incoming events and use a router module to decide whether to process the event on that device. If the event is unable to be processed at the current level, a pusher module is designed to push the event to the upper level for that layer to make the decision and propagate similarly. Their solution makes addition and removal of tasks a local decision made by the router modules, therefore making the task placement decision on the device based on local knowledge only.

2.6 Evaluation Tools

With the increasing amount of strategies presented for utilising edge computing resources in the literature, the platforms used for testing these strategies have also become useful contributions. We have identified two main categories of evaluation tools based on their contrasting characteristics, namely physical testbeds, and simulators.

2.6.1 Physical Testbeds

Physical experimental setups or testbeds introduce the intricacies and complexities of the real world into the experiments with a certain degree of control available for the experiment designers while also being the most practical approach. As a result, experiments run on physical setups tend to present the most realistic conclusions.

Grid'5000 [209] is a physical testbed that provides 8 sites, 31 clusters, 828 nodes with 12328 cores spread mainly in France with a dedicated 10 Gbps backbone network designed for researchers as well as developers to conduct high quality, reproducible, large-scale experiments on distributed and parallel computing as utilised in [210], [211], [78], and [212]. Grid'5000 is now merged with the FIT (Future Internet Testing facility) consortium. FIT IoT-LAB [213] is another testbed that consists of 2728 low power wireless nodes and 117 mobile robots, which is specifically designed for IoT related experiments which is now widely being used for edge and cloud based realistic experiments as presented in studies such as [78] and [214]. Cumulus [215] is another edge computing platform with multiple mobile devices, single board computers and desktop computers. Apart from these platforms ad-hoc testbeds are also used in the literature as shown in the works of [1], [19], [80], [132], [85], [133], [88], [99], [134], [135], [190], and [136] where single board computers such as Raspberry Pis are being used as edge devices which are then connected through a WLAN and to cloud virtual machines or containers via a WAN if necessary. Notably, Pahl et al. [137] have developed a container based edge computing platform on Raspberry Pi devices which extends the functionality of edge resources as a platform as a service. In Mobistreams, Wang and Peh [216] use 8 iPhones connected to base stations and each other via WiFi and 3G connections, as their experimental setup for MEC. Android based mobile phones connected via WiFi and LTE networks are used as edge devices by Ogden and Guo [217], and Dolezal, Becvar and Zeman [218]. However, it is important to identify that while physical testbeds provide realistic results for practical applications, there exist two main drawbacks of implementing testbeds which are the financial and time cost of devising a comprehensive computing infrastructure to fully capture the heterogeneity of real edge devices as well as the concerns on reproducibility of the experiments due to uncontrollable physical intricacies.

2.6.2 Simulators

To overcome some of these disadvantages of physical testbeds, simulation tools are implemented or utilised in state-of-the-art where various device configurations and application scenarios are imitated using software.

CloudSim [219] is a simulation toolkit designed for simulating system components as well as resource provisioning policies for cloud environments. Even though CloudSim does not support edge or fog computing resources, it has been extended by iFogSim [92] to add support for simulating IoT data streams and devices. The authors of iFogSim have added support for IoT resource management, monitoring, power management as well as support for application models where these features are readily used in [81], [69], [220], [82], [221], [63], [97], [222], [105], [223], [224], [191], [104], and [68]. EdgeCloudSim [225]

is a similar simulation toolkit built on top of CloudSim to add support for a multi-tier networking model and computational resources for edge-specific simulations and is used in [189] and [226].

OMNeT++ [227] is a well established network simulation tool that has realistic network protocol support with the help of the INET framework. However, while studies such as [228] and [74] utilise OMNeT++ for simulations, one of the main drawbacks of OMNeT++ is the lack of processing simulation support. As a result, OMNeT++ is mainly used as a platform for building simulation tools where various computational support is added depending on the type of application and the device configurations. CloudNetSim [229] and CloudNetSim++ [230] are such simulation environments built on top of OMNeT++ while focusing more towards cloud computing resources such as data centres. FogNetSim++ is another simulation environment built on top of OMNeT++ that supports implementing large scale fog networks by adding task scheduling support, device handover and mobile computing nodes while leveraging the network simulation support inherent in OMNeT++. NS-2 simulator [231] is another legacy network simulator which is being used by Morales, Rosas and Hidalgo [232], and Munir, Antoon and Gordon-Ross [233] as the base for their simulations.

In addition to the above mentioned utilisations of established platforms, ad-hoc simulation solutions are also abundant in the literature. FogTorchΠ [119] and FogTorch [126] are Monte-Carlo simulators built for QoS (latency and bandwidth) aware task placement in heterogeneous resources. De Maio and Brandic [76] extend FogTorchΠ by adding support for mobility, power consumption and DAG based task scheduling. Ascigil et al. [154] and [192] utilise the Icarus network simulator [234] whereas Ouyang, Zhou and Chen [186] utilise the ONE simulator [235]. In [93], the authors utilise the Simgrid simulator. MATLAB is also used for developing simulations in studies such as [160], [159], [165], [167], and [236]. Furthermore, ad-hoc simulations are presented in [103], [237], [238], [239], [75], [162], [240], [241], [187], [242], [174], [98], [179], [243], and [244].

The key advantages of simulators are the ability to configure and control the devices, applications and the overall experiment which makes it possible to share these configurations which leads to easily reproducible experiments, the lower cost of implementing a comprehensive and heterogeneous device configurations, and the extendibility of the system to add device and application configurations in the future.

On the other hand, it is not always possible to fully mimic a physical testbed with a simulator and as a result the generated results need to be verified against a real setup if applicable. We have noted that none of the existing state-of-the-art simulators have been verified against a real testbed.

2.7 Summary

The effect of Internet of Things and Edge Computing is undeniable in the modern information age where it is considered that more information is processed, the better. For processing continuous streams of events, generated by billions of devices distributed in our households and in the most remote parts of the world, Data Stream Processing paradigm has been introduced which conducts in-memory processing of incremental data. Combining data stream processing with the potential of the edge computing has been of great interest for the industry as well as the academia. Therefore, identifying the characteristics of this amalgamation and devising strategies presented by the state-of-the-art in this domain for sharing the workload of DSP applications, defined as task placement, between the already established cloud computing and emerging edge computing has been the focus of this chapter. We have presented a comprehensive taxonomy to the state-of-the-art in this front, ranging from the earlier approaches in the cluster computing to the deep learning based task placement for stream processing engines. We have discussed in detail the characteristics of each relevant work in each category and compared their strengths and weaknesses. In order to cover some of the gaps in research identified in this literature survey, our research contributions to data stream processing on edge-cloud environments are presented in the following chapters.

Chapter 3

A Model-based Simulator for Distributed Data Stream Processing

This chapter is derived from:

- **Gayashan Amarasinghe**, Marcos Dias de Assunção, Aaron Harwood, and Shanika Karunasekera, "ECSNeT++ : A simulator for distributed stream processing on edge and cloud environments," *Future Generation Computer Systems*, vol. 111, pp. 401–418, 2020.

3.1 Introduction

Devices are rapidly becoming connected as the recent surge of Internet of Things (IoT) has led to widespread introduction and usage of smart apparatus in households as well as commercial environments. As a result, the global IoT market is expected to hit $7.1 trillion in 2020 [245]. These smart devices have demonstrated to hold great potential to improve many sectors such as healthcare, energy, manufacturing, building construction, maintenance, security, and transport [246][247][210][248]. For example, Trenitalia, the main Italian train service operator is currently using an IoT based monitoring system to move from a reactive maintenance model to a passive maintenance model to deliver a more reliable service to travellers [249].

One common trait of many IoT deployments is that the devices monitor the environment they are deployed in, and produce events related to the current status of the said

environment. This generates a time series of events that can be processed to extract vital information about the environment. Generally, this event stream is unbounded and hence, DSP techniques can be utilised in order to process the events in real-time [43].

Apache Storm [50], Twitter Heron [51], Apache Flink [52] are modern DSP engines widely used for distributed stream processing. Cloud computing has become the main infrastructure for deploying these engines, where they are generally hosted by large-scale clusters offering enormous processing capabilities and whose resources are accessed on demand under a pay-as-you-go business model.

As mentioned earlier, with the involvement of IoT, the sources where events are generated are located outside the cloud. Therefore, if we are to process these event streams using DSP engines, the current status quo is to generate data streams at the IoT devices and transmit them to the cloud for processing [43]. However, this is not ideal as the higher latency caused by the communication overhead between IoT devices and the cloud often becomes a bottleneck [36][34].

On the other hand, *Edge computing* is emerging as a potential computing paradigm that can be utilised for sharing workloads with the cloud for suitable applications [250]. The goal of edge computing is to bring processing closer to end users or in proximity to data sources [34][39]. As a result, concepts such as *cloudlets*, *Mobile Edge Computing*, and *Fog Computing*, are utilised where the processing is conducted at the base stations of mobile networks or servers that reside at the edge of the network infrastructure, sometimes even referred to as the *Edge Cloud* [39]. However, with the advancements in processing and networking technologies, processing should not be limited only to these processing servers that are at the edge of the network infrastructure. Therefore, we utilise a broad definition of Edge Computing as computations executed on any processing element, edge servers and/or clients, located logically outside the cloud [34][39][251][61][6]. These devices are located closer to the data sources, sometimes acting as data sources or thick clients themselves. Wearable devices, mobile phones, personal computers, single board computers, many common smart peripherals with computational capabilities (such as many IoT devices), mobile base stations, and networking devices with processing capabilities belong to this category. With the increasing computational power of mobile processors, we believe these *edge devices* located closer to the end users, have potential that is not being fully utilised [232].

The commercial interest in the utilisation of IoT technologies for processing data at the edge, is shown with the invention of Google's implementation of artificial intelligence at the edge [6], in-device video analysis devices [9][10][11], home automation devices [12], and industrial automation applications [13]. However, these edge devices have

limitations of their own as well – such as being powered by batteries and being connected to the cloud with limited bandwidth networks, which are often wireless.

The number of Edge computing devices (including IoT devices) is increasing exponentially. These devices have heterogeneous specifications and characteristics which cannot be generalised in a trivial manner. Many of these devices have potential that can be utilised in many ways. Therefore, finding efficient ways to utilise these heterogeneous resources is important and a timely requirement. However, creating a real test environment for running such experiments is challenging due to the amount of effort and investment required. As a result, simulation has become a suitable tool that can be used to run these experiments and to acquire realistic results [92][225].

Currently, there is no complete simulation environment that provides both an accurate processing model and a networking model for edge computing applications – specifically for distributed stream processing application scenarios. The available simulation tools lack an accurate network communication model [92][225]. However, on edge computing applications, the network communication plays a critical role mainly due to the bandwidth and power limitations [225]. In addition, none of the available simulation tools have been verified against a real device deployment which we believe is equally important since simulation without a sense of accuracy and reliability of the acquired results is ineffective.

As a solution, we have implemented a simulation toolkit, ECSNeT++, with an accurate computational model for executing DSP applications, as an extension to OMNeT++, which is one of the widely used network simulators due to its in-detail network modelling and accuracy [225][227]. We have verified ECSNeT++ against a real device deployment to ensure that it can be properly calibrated to acquire realistic measurements compared to a real environment.

More specifically, we make the following contributions:

- We develop a comprehensive model that defines the characteristics of data stream processing applications running on edge and cloud computing infrastructure.
- Based on our model, we present an extensible simulation toolkit, which we call ECSNeT++, built on top of the widely used OMNeT++ simulator that is able to simulate the execution of DSP application topologies on various edge and cloud configurations to acquire the end-to-end latency, processing latency, network latency, and per-event energy consumption measurements.

- We demonstrate the effectiveness and adaptability of our simulation toolkit by simulating multiple configurations of two realistic DSP applications from the RIoTBench suite [1] on edge-cloud environments.
- We verify our simulation toolkit against a real edge and cloud deployment with multiple configurations of the two RIoTBench DSP applications to show that the measurements acquired by simulations can closely represent real-world scenarios with proper calibration.
- We display the extensibility of our simulation toolkit by adopting the LTE network capability from the SimuLTE framework [252] to simulate a realistic DSP application on an LTE network.
- We have made our simulation toolkit[1] and the LTE plugin[2], open source and publicly available for use and further extension.

A high-level comparison of the features of ECSNeT++ against the state-of-the-art simulators publicly available as shown in Table 3.1, indicates that ECSNeT++ is the only simulation tool of its kind for simulating the execution of DSP applications on edge and cloud computing environments that has also been verified against a real experimental set-up to demonstrate its capabilities.

TABLE 3.1: Comparison of ECSNeT++ simulation toolkit against state-of-the-art.

Feature	CloudSim [219]	EdgeCloudSim [225]	OMNeT++ [227]	iFogSim [92]	ECSNeT++ toolkit
Edge Devices	×	✓	×	✓	✓
Cloud Server	✓	✓	×	✓	✓
Realistic Network Model	Limited	✓	✓	Limited	✓
Power Model	✓	×	✓	✓	✓
Processing Model	✓	✓	×	✓	✓
Device Mobility	×	✓	✓	×	✓
Comprehensive DSP Task Model	×	×	×	Limited	✓
DSP Task Placement	×	×	×	✓	✓
Verification against real set-up	×	×	×	×	✓

The rest of this chapter is organised as follows. We provide a brief background about the characteristics of host networks and DSP applications in Section 3.2, whereas the modelling and simulation tool is described in Section 3.3. The evaluation scenario and performance results are presented in Section 3.4. Section 3.5 discusses related work and Section 3.6 concludes the chapter.

[1] Available at https://sedgecloud.github.io/ECSNeTpp/
[2] Available at https://github.com/sedgecloud/ECSNeT-LTE-Plugin

3.2 Background

3.2.1 Characteristics of a Host Network

An execution environment (i.e. a host network) is a connected graph where vertices represent devices and the edges of the graph are the network connectivity. In our study we consider an environment that consists of cloud servers and edge computing devices, where the edge devices are connected to the cloud through a LAN and WAN infrastructure.

Each device has data processing characteristics (as defined by the available CPU and the associated instruction set), network communication characteristics (according to the Network Interface Controller and network protocols), and data storage characteristics (such as volatile and non-volatile storage). The network itself has a topology, different network components such as routers, switches and access points, and quantifiable characteristics such as the network latency, bandwidth and throughput.

The processing power and storage characteristics vary depending on the host device. A cloud node usually provides a virtual execution environment often with high processing power with multi-core multi-threaded CPUs, and high RAM capacity and speed. In contrast, an edge device has lower processing power and memory. For example, a Raspberry Pi 3 Model B device has a 4 core CPU with 1024 MB memory, while a high-end mobile phone may have a quad/octa-core CPU with 6/8 GB memory. In addition, these different devices may have their own instruction sets depending on the processor architecture. All of these processing characteristics and memory characteristics are generally available with the device specifications.

The most power consuming components of these host devices are the network subsystem and the processing unit. The power consumption characteristics of hosts for the most part depend on the underlying hardware. The cloud servers consume significant amount of power since they have unconstrained access to power. On the other hand, the edge devices are constrained by power availability. However, in most of these devices, it is possible to monitor and profile the power consumption characteristics empirically [253] or acquire them from the device specifications.

The characteristics of the underlying network depend on the network topology and the components of the network. A Local Area Network has higher bandwidth and throughput with lower latency but a smaller, less intricate topology. A Wide Area Network has a more complex topology but often with higher latency that increases with distance. Depending on the location of the edge devices and cloud servers, a combination of LAN and WAN may be utilised to build the host network.

3.2.2 Characteristics of a DSP Application

As illustrated in Figure 3.1, a DSP application consists of four components, *source*, *operator*, *sink*, and the *dataflow*. In any DSP application, the sinks, sources and operators, known as *tasks*, are placed and executed on a processing node. These different tasks in the application topology have their own characteristics depending on the application.

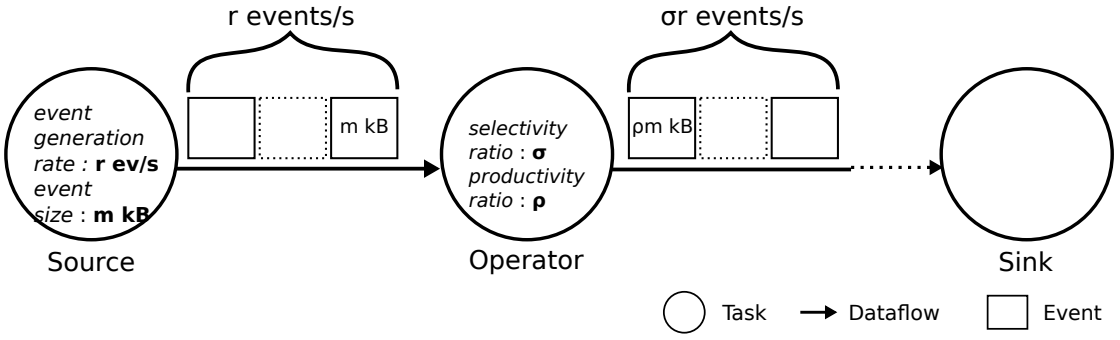


FIGURE 3.1: Components and characteristics of a DSP application where a source generates events, which are then processed by the operators down stream, and reach a sink for final processing. Each source governs the event generation rate and the size of each event generated. Each operator manipulates either the properties of each event, the event processing rate or both.

A source generates events – therefore has an *event generation rate* and an *event size* associated with it. The event generation rate vary from one application to another. We can expect a uniform rate if the source is periodically reading a sensor and transmitting that value to the application. If the source is configured to read a sensor and report only if there is a change in reading, it can produce events at a rate that is difficult to predict. If the source is related to human behaviour, e.g.: activity data from a fitness tracker, the event generation rate can follow a bimodal distribution with local maxima, where people are most active in the morning and the evening [1].

Each generated event has a fixed or variable size depending on the application. If the event is generated by reading a specific sensor, often an upper bound for the event size is known due to the known structure of an event. For example, a temperature reading from a thermometer may contain a timestamp, sensor ID, and the temperature reading. Therefore, the size of such an event can be trivially estimated. However, if each event is a complex data structure such as a twitter stream, which can contain textual data and other media, then the event size may vary accordingly.

For a given operator, two characteristics control the dataflow. For an operator T_j, the *selectivity ratio* σ_{T_j} is the ratio between the outgoing throughput $\lambda^{out}_{T_j}$ and the ingress throughput $\lambda^{in}_{T_j}$ [241][1], *i.e.* $\sigma_{T_j} = \lambda^{out}_{T_j}/\lambda^{in}_{T_j}$. We also introduce, for an operator T_j, *productivity ratio* ρ_{T_j}, which is the ratio between an outgoing message size $\beta^{out}_{T_j}$ and the incoming message size $\beta^{in}_{T_j}$, *i.e.* $\rho_{T_j} = \beta^{out}_{T_j}/\beta^{in}_{T_j}$. The distribution of the selectivity and

productivity ratios depend on the incoming event size and the transformation conducted at the operator [1]. For example, if the operator is calculating the average of incoming temperature readings for a given time window, an upper bound for the productivity ratio can be estimated since outgoing event structure is known. If the window length is known, the selectivity ratio can also be estimated. However, if the operator is tokenizing a sentence in a textual file, the productivity ratio and the selectivity ratio for that particular operator can vary depending on the input event. By profiling each operator, the selectivity and productivity ratio distribution can be estimated to a certain extent. But this is highly application dependent.

3.3 ECSNeT++ Simulation Toolkit

ECSNeT++ is a toolkit that we designed and implemented to simulate the execution of a DSP application on a distributed cloud and edge host environment. The toolkit was implemented on top of the OMNeT++ framework[3][227] using the native network simulation capabilities of the INET framework[4]. OMNeT++ framework has matured since its inception in 1997 with frequent releases[5]. In addition to its maturity and widespread use among the scientific community[229][230], OMNeT++ enables the development of a modular, component-based toolkit to simulate the inherent characteristics of edge-cloud streaming applications and networks.

The high-level architecture of ECSNeT++ is illustrated in Figure 3.2. We implemented the host device model, and the power and energy model by extending the INET framework. The multi-core multi-threaded processing model is built on top of our host device model. The components of the DSP application model use all these features to simulate the execution of an application on an Edge-Cloud environment. The following subsections describe in detail how the different elements of a stream processing solution are modelled in the simulation toolkit. In Table 3.2 we summarize the notation used throughout this study.

3.3.1 Host Network and Processing Nodes

Although our approach is not limited to two logical computational layers of edge and cloud, for this study we have only focused on two layers. Therefore, we have implemented two categories of processing nodes: cloud nodes and edge nodes. As mentioned earlier, the edge computing nodes are the leaf nodes that are connected to the network via

[3] https://omnetpp.org/
[4] https://inet.omnetpp.org/
[5] OMNeT++ v5.3 was released on April, 2018.

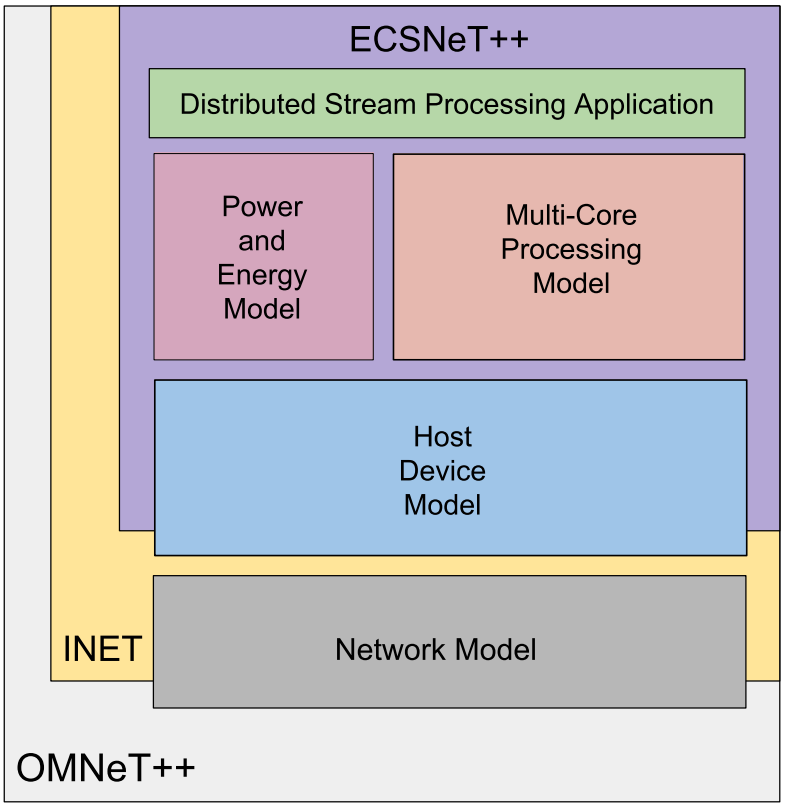


FIGURE 3.2: The high level architecture of ECSNeT++ which is built on top of the OMNeT++ simulator and the INET framework.

an either a IEEE 802.11 WLAN using access points for each separate network or via an LTE Radio Access Network with the use of ECSNeT++ LTE Plugin and eNodeB base stations. Depending on the scalability of the edge layer, the number of access points or base stations can be configured. They act as the bridge between the edge clients and the wired local area network (See Figure 3.4 and Figure 3.17). All the wired connections are simulated in this host network as full duplex Ethernet links. However, by utilising the INET framework with our host device implementations, it is possible to simulate a complex LAN and WAN configuration as required. In addition, the use of an LTE network for communication between the edge and the cloud by using the SimuLTE simulation tool [6] and the ECSNeT++ LTE Plugin, which provides a LTE network connectivity model, shows the extensibility of the networking model available in ECSNeT++.

We have implemented each type of edge device that use IEEE 802.11 connectivity by extending the `WirelessHost` module of INET framework which simulates IEEE 802.11 network interface controllers for communication. For LTE connected edge devices, each device is a stand-alone module with a LTE enabled network interface controller which implements the `ILteNic` interface from the SimuLTE tool. A cloud node is implemented

[6] http://simulte.com/index.html

Table 3.2: Notations used in the modelling of ECSNeT++.

Symbol	Description
T_i	Task i
Op_i	Operator i
So_i	Source i
Si_i	Sink i
V_j	Edge device or Cloud server j
σ_{T_i}	Selectivity ratio of T_i
ρ_{T_i}	Productivity ration of T_i
r_{So_i}	Event generation rate of So_i
m_{So_i}	Generated event size of So_i
$\delta_{T_iV_j}$	Processing time per event, per-task T_i, per-device V_j
$\zeta_{T_iV_j}$	CPU cycles consumed per event, per-task, per-device
C_{V_j}	Per core processing frequency of device V_j
η_{V_j}	Number of CPU cores on device V_j
ψ_{V_j}	Number of threads per CPU core on device V_j
Φ_{total}	Total power consumption
Φ_{idle}	Idle power consumption
$\Phi_{cpu}(u)$	CPU power consumption at utilisation u
$\Phi_{wifi,idle}$	WiFi idle power consumption
$\Phi_{wifi,up}$	WiFi transmitting power consumption
$\Phi_{wifi,down}$	WiFi receiving power consumption

by extending the `StandardHost` module of the INET framework. This grants the cloud nodes the ability to connect to other modules (switches, routers, etc.) via a full-duplex Ethernet interface.

Each host has the capability to use either TCP or UDP transport for communication. The preferred protocol can be configured by enabling the `<host>.hasTcp` or `<host>.hasUdp` property and setting the `<host>.tcpType` or `<host>.udpType` property appropriately on the `omnetpp.ini` file (or the relevant simulation configuration file) to use the TCP or UDP protocol for communication.

3.3.2 DSP Application

Three separate task modules were developed to represent namely sources, operators, and sinks. A `StreamingSource` task module produces events depending on the configurations for event generation and event size. A `StreamingOperator` task module accepts incoming events, processes them using the processing model of the host and sends them to the next task or tasks according to the DSP topology. A `StreamingSink` task module is the final destination for all incoming events.

Each of these task modules should be placed on a host device with processing capabilities and can be connected according to the adjacency matrix of the DSP topology, which

needs to be provided to the simulator, to simulate the dataflow of a DSP application. Instances of these modules should be configured, depending on the characteristics of each task in the DSP application topology. The processing of events in each of these tasks shall be done according to the processing model we have implemented in the simulation toolkit.

3.3.2.1 Placement Plan

The placement plan, which dictates the placement of one or more DSP application topology tasks (Sources, Operators and Sinks) on to the host networks, needs to be generated externally depending on the requirements of the application deployment and should be submitted to ECSNeT++. For example, the placement plan contains a mapping for each task in the DSP application topology to a host device (or devices) along with other characteristics relevant to the task if it is executed on the particular host device (or devices). Therefore, along with the placement of the topology, every attribute related to each task, e.g. the processing latency of each task, event generation rate and generated message size of a source, selectivity and productivity distribution of an operator, need to be configured in the placement plan itself. We have implemented an XML schema to define the placement plans. We have decoupled the placement plan generation from the ECSNeT++ to allow the users to use any external tool to generate placement plans and to produce an XML representation of the plan using our schema.

3.3.2.2 Source

`StreamingSource` module represents a streaming source in the topology. Responsibility of each `StreamingSource` module is to generate events as per the configuration and to send the generated events downstream to the rest of the application topology. Two attributes affect the event stream generated at any given source in the topology; the generated event size and the distribution of the event generation rate. These attributes can be configured in the `StreamingSource` module. Once configured, the `StreamingSource` module starts generating events as per the message size and event rate distributions and sends the generated events to the adjacent tasks.

The generated event size depends on the type of the application topology. A developer can simulate this characteristic by implementing the `ISourceMessageSizeDistribution` interface and configuring the placement plan by assigning the characteristic to the respective source such as a uniformly distributed message size distribution. Similarly

the `ISourceEventRateDistribution` interface can be implemented, followed by an appropriate configuration in the placement plan, to incorporate different event generation rates for each source in the topology such as a bimodal event generation rate distribution.

Once these characteristics are configured for each source, they will produce events accordingly, that can be processed by operators connected to the data flow.

3.3.2.3 Operator

An operator in a topology can be simulated by configuring and placing a `StreamingOperator` module. The responsibility of each operator is to process any and all incoming streaming events and to produce one or more resulting events based on the characteristics that define its function; the selectivity ratio and the productivity ratio. By implementing the `IOperatorSelectivityDistribution` interface, the selectivity ratio distribution of each operator can be simulated. Similarly the `IOperatorProductivityDistribution` interface should be implemented to simulate the productivity ratio distribution of a respective operator. Each of these implementations needs to be configured with appropriate parameters according to the application topology in the placement plan. The parameter values can be acquired by profiling the incoming and outgoing events of each operator or by analysing the topology.

When an event arrives at an operator, the operator changes the outgoing event size and the outgoing event rate based on the productivity distribution and the selectivity distribution of that operator, and emits a corresponding event (or events) which are sent to the next operator or sink according to the application topology (denoted by the adjacency matrix in the simulator).

3.3.2.4 Sink

We have implemented the `StreamingSink` module to simulate the functions of a sink in the topology. Each sink acts as the final destination for all the arriving events and emits the processing latency, network latency, and the total latency of each event.

Apart from the above configurations, for each of these sources, operators, and sinks, the attributes to calculate processing latency needs to be configured. The following subsection describes how the processing model has been implemented and how to configure this model appropriately for each task in the DSP application topology being simulated.

3.3.3 Processing Model

Each processing node in the host network has a processing module to simulate a multi-core, multi-threaded CPU. The processing of an event is simulated by holding the event in a queue for a time period $\delta_{T_i V_j}$.

Each processing node shall be configured with 3 parameters; number of CPU cores η_{V_j}, frequency of a single CPU core C_{V_j}, and the number of threads per core ψ_{V_j}. The `CPUCore` module of ECSNeT++ is implemented to simulate a single CPU core instance. Each host device has `CPUCore` module instances equal to the number of CPU cores in that particular host as configured.

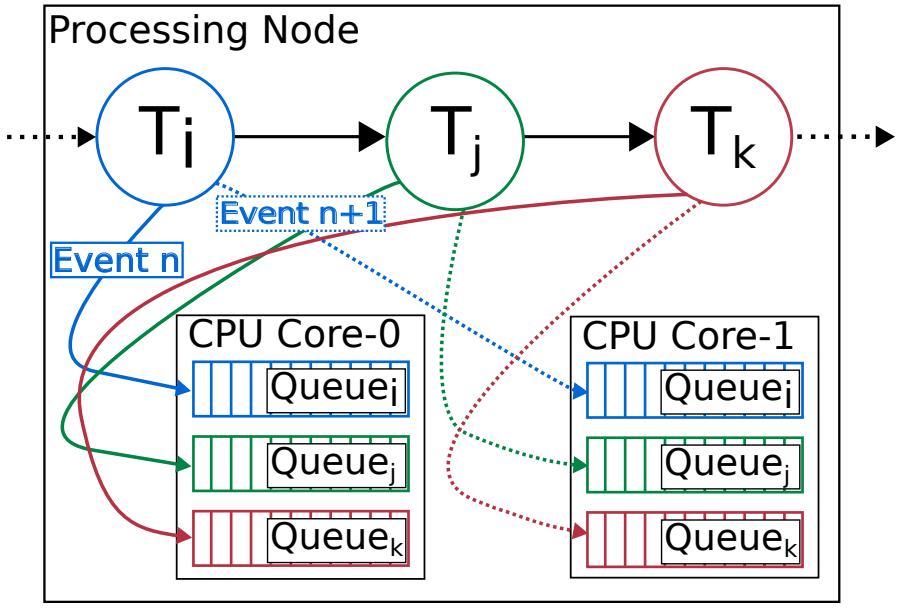


FIGURE 3.3: A processing node (V_a) with a dual core CPU and 3 DSP tasks (T_i, T_j, T_k) assigned to it, processes an incoming event n by sending the event to a selected CPU core in round-robin manner. Each CPU core maintains an exclusive non-priority queue for each task. Each queue holds an unprocessed event for a time period $\delta_{T_i V_a}$ and releases the event back to the task that sent the event to that particular CPU core. This concludes processing of the particular event by the said task.

Figure 3.3 shows the processing of an event at a processing node. Since each processing node has tasks placed on it according to the placement plan as described earlier in Section 3.3.2.1, when an event arrives at a particular task, the task selects a CPU core for processing depending on the scheduling policy. We implemented a *Round-robin scheduler* for this study. However, it is possible to extend this behaviour to introduce other scheduling policies to the processing model by implementing the `ICpuCoreScheduler` interface. Once a CPU core is selected, the task sends the event to that core for processing. A CPU core maintains an exclusive non-priority FIFO queue for each operator, simulating the behaviour of a multi-threaded execution where the execution of each task is done in a single thread which depicts a thread per task architecture.

There exist two methods to acquire the per event processing latency for a task running on a particular device ($\delta_{T_iV_j}$). Each task can be configured in the placement plan to directly provide per event processing time which can be measured by profiling the running time of a task per event. If this is not feasible, the number of CPU cycles required to process the particular event in the particular CPU architecture can be determined empirically by profiling an operator [254] or even by analysing the compiled assembly code, depending on the complexity of the application code, which in turn can be used to calculate the per event processing time.

In our experiments we use per event processing time directly since it can be measured from our real experiments.

After holding the event in the queue for the processing latency acquired by either of the above mentioned methods, the CPU core removes the event from the queue, and sends it back to the task that sent it, which completes the processing of the particular event on that task.

When a source has generated an event or an operator receives a processed event, it sends the event to the next downstream operator or sink. Each event is processed in this manner until it reaches a sink. This processing model is used by all the sources, operators, and sinks in the configured DSP application during the simulation.

3.3.4 Power and Energy Model

In ECSNeT++ we have implemented two power consumption characteristics for the edge devices connected via the IEEE 802.11 WLAN. The network power consumption measures the amount of power consumed while receiving and/or transmitting a processed event. The processing power consumption measures the amount of power consumed while processing a particular event at a CPU core. In addition, the idle power consumption measures the amount of power consumed by the device when idle.

We have implemented and configured the `NetworkPowerConsumer` class to simulate the power consumption of the network communication over IEEE 802.11 radio medium. Each processing node in the edge layer has an instance of a `NetworkPowerConsumer` assigned to it. When the state of the radio changes, the consumer calculates the amount of power that would be used by that state change and emits that value. The `NetworkPowerConsumer` class has to be configured with idle power consumption of the device without any load, idle power consumption of the network interface, transmitting power consumption of the network interface, and the receiving power consumption of the network interface. These values can be measured or estimated based on empirical data

[253][255]. The power consumption is determined, depending on the radio mode, the state of the transmitter, and the state of the receiver.

We have implemented and configured the `CPUPowerConsumer` class on ECSNeT++ to simulate the power consumed by the processor in each edge based processing node. With our processing model, a CPU core operates within two states, CPU idle state and the busy state. When the CPU core is processing an incoming event, as per Section 3.3.3, it operates in the busy state and otherwise in an idle state. When a CPU core changes its state, the `CPUPowerConsumer` calculates the amount of power associated with the relevant state change. The `CPUPowerConsumer` is configured using the power consumption rate with respect to its utilisation which can be measured or estimated [253][255].

When a power consumption change is broadcast by a consumer, the `IdealNodeEnergyStorage` module calculates the amount of energy consumed during the relevant time period by integrating the power consumption change over the time period and records that value. However, it is important to note that the `IdealNodeEnergyStorage` does not have properties such as memory effect, self-discharge, and temperature dependence that can be observed on a real power storage.

The total power Φ_{total} consumed at each edge device can be calculated by the following Equation (3.1) [253][255]. The total power consumption is estimated by summing the power consumption of individual components as configured. Here Φ_{idle} is the idle power consumption without any load on the system, $\Phi_{cpu}(u)$ is the power consumed by CPU when the utilisation is $u\%$, and $\Phi_{wifi,up}(r)$ and $\Phi_{wifi,dn}(r)$ represent the power consumed by the network interface for transmitting and receiving packets over the IEEE 802.11 WLAN at the data rate of r.

$$\Phi_{total} = \Phi_{idle} + \Phi_{cpu}(u) + \Phi_{wifi,idle} + \Phi_{wifi,up}(r) + \Phi_{wifi,dn}(r) \tag{3.1}$$

The components of this approximation function can be calibrated in the simulation configuration by either measuring the individual power consumption components of the device(s) or by referring to power characteristics specifications of the device(s). Both uplink and downlink energy consumption should be considered since as per our state based power model and the duplex nature of packet transmission, arrival or the departure of a packet at the relevant network interface, triggers the energy consumed for that state change at the network interface to be recorded. All such recorded state changes are accumulated to produce the total power and energy consumption for the simulation.

TABLE 3.3: Specification of the edge and cloud devices used in experimental set-up used for verification of ECSNeT++.

	Edge	Cloud
Proc. Architecture	ARMv7	x86_64
Processor Freq.	1.2GHz	2.6GHz
CPU Cores	4	24

As we are only interested in the power/energy consumption of the edge devices, we have limited the power consumption simulation to only the edge devices, and the wireless network communication between the edge devices and the access points. However, due to the modular architecture of the ECSNeT++, the power model can be extended to other processing nodes and communication media such as the LTE RAN as well.

Even though we only consider cloud and edge nodes placed in two logical layers, it is possible to create hierarchical layers of cloud and edge nodes by extending either the `StandardHost` module or the `WirelessHost` module depending on the connectivity requirements and then by adding relevant processing model and power consumption model depending on the specifications of the device. ECSNeT++ supports placement of tasks on any connected node with CPU modules for processing.

3.4 Experiments and Evaluation

We use two set-ups for experiments. To run the ECSNeT++ simulator in headless mode we use a m2.xlarge instance, which consists of 12 VCPUs, 48GB RAM, and a 30GB primary disk on the Nectar cloud[7]. We execute the simulation of ETL and STATS application topologies on this set-up with different operator allocation plans, and acquire the relevant measurements.

In order to verify the IEEE 802.11 WLAN based simulations of ECSNeT++ simulator against a real-world experimental set-up, we have established a simple experimental set-up using 2 Raspberry Pi 3 Model B (edge) devices which connect to two access points to create 2 separate edge sites. The access points are then connected to a router which connects to a single cloud node (cloud). The cloud node has the processing power of 24 core Intel(R) Xeon(R) CPU E5-2620 v2 at 2.60GHz and 64GB of memory. This selected host network for the real experiment set-up is shown in Figure 3.4 and the configurations are available in Table 3.3. The clocks of all these host devices are synchronized using the Network Time Protocol. All the communications between the edge and the cloud occur over the TCP protocol.

[7] https://nectar.org.au/research-cloud/

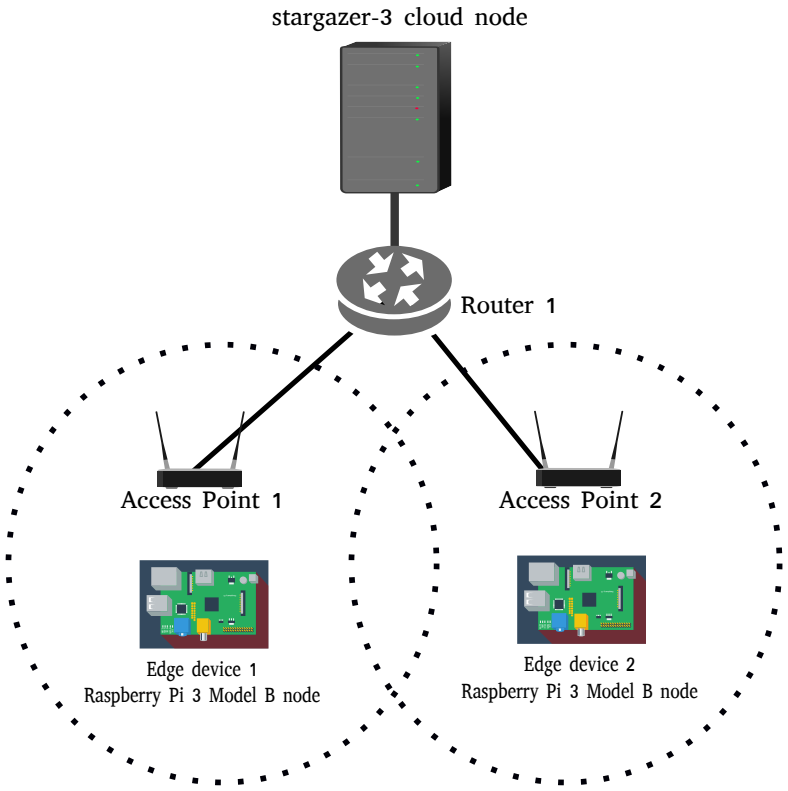


FIGURE 3.4: Experimental set-up used for simulator calibration and verification.

We have implemented the DSP applications using Apache Edgent [57] framework which runs on Java version 1.8.0_65 on each edge device and cloud server in the experimental set-up. Apache Edgent is a programming model and a micro kernel which provides a lightweight stream processing framework that can be embedded in many platforms.

3.4.1 Application Topologies and Allocation Plans

We use a simplified version of the ETL application topology and the STATS application topology from the RIoTBench suite [1] as the DSP applications for the experiments. The ETL application has a linear topology (See Figure 3.5) which extracts information from a given CSV file and converts the information to SenML format after preprocessing. The STATS application has a diamond topology (See Figure 3.6) which extracts information from a given CSV file and runs a statistical analysis of the information to produce a set of line plots. These topologies can be executed on either, edge, cloud or both.

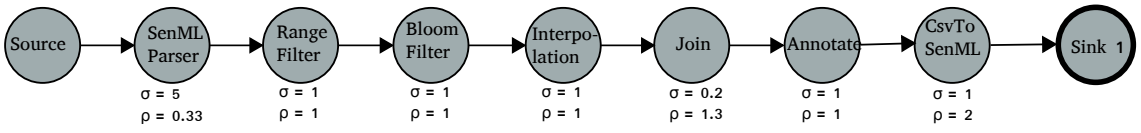


FIGURE 3.5: Simplified ETL application topology from the RIoTBench suite.

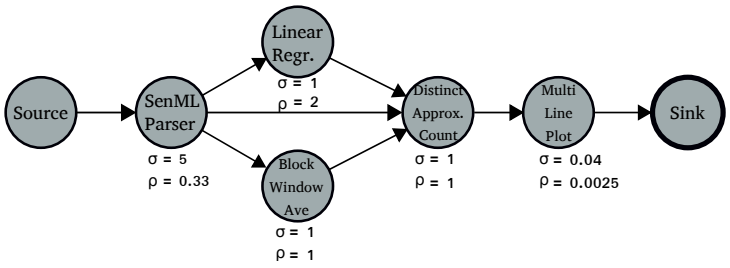


FIGURE 3.6: Simplified STATS application topology from the RIoTBench suite.

TABLE 3.4: Characteristics of the operators of the ETL and STATS applications used in the experiments.

Application	Operator(s)		
	Name	Selectivity	Productivity
ETL	SenML Parser	5	0.33
	Range Filter	1	1
	Bloom Filter	1	1
	Interpolation	1	1
	Join	0.2	1.3
	Annotate	1	1
	Csv To SenML	1	2
STATS	SenML Parser	5	0.33
	Linear Regression	1	2
	Block Window Average	1	1
	Distinct Approx. Count	1	1
	Multi Line Plot	0.04	0.0025

We have partitioned the ETL application topology into two partitions in 7 ways as illustrated in Figure 3.7 where one partition is placed on the edge device (purple nodes) and the other partition is placed on a cloud server (blue nodes). The two partitions communicate using the communication media as configured. Then we execute the application on both the real world set-up and the simulator to process a fixed number of events, and acquire latency and edge energy consumption measurements. Similarly, we have partitioned the STATS application topology into two partitions in 6 ways where one partition is placed on the edge and the other is placed on the cloud. Then we execute each placement plan on both the simulator and the real experiment set-up. For each plan, we also vary the source event generation rate while keeping other parameters constant.

3.4.2 Measurements

In order to evaluate the applications, we are using four measurements – processing latency per event, network transmission latency per event, total latency per event, and average power consumption at each edge device. These measurements can be collected

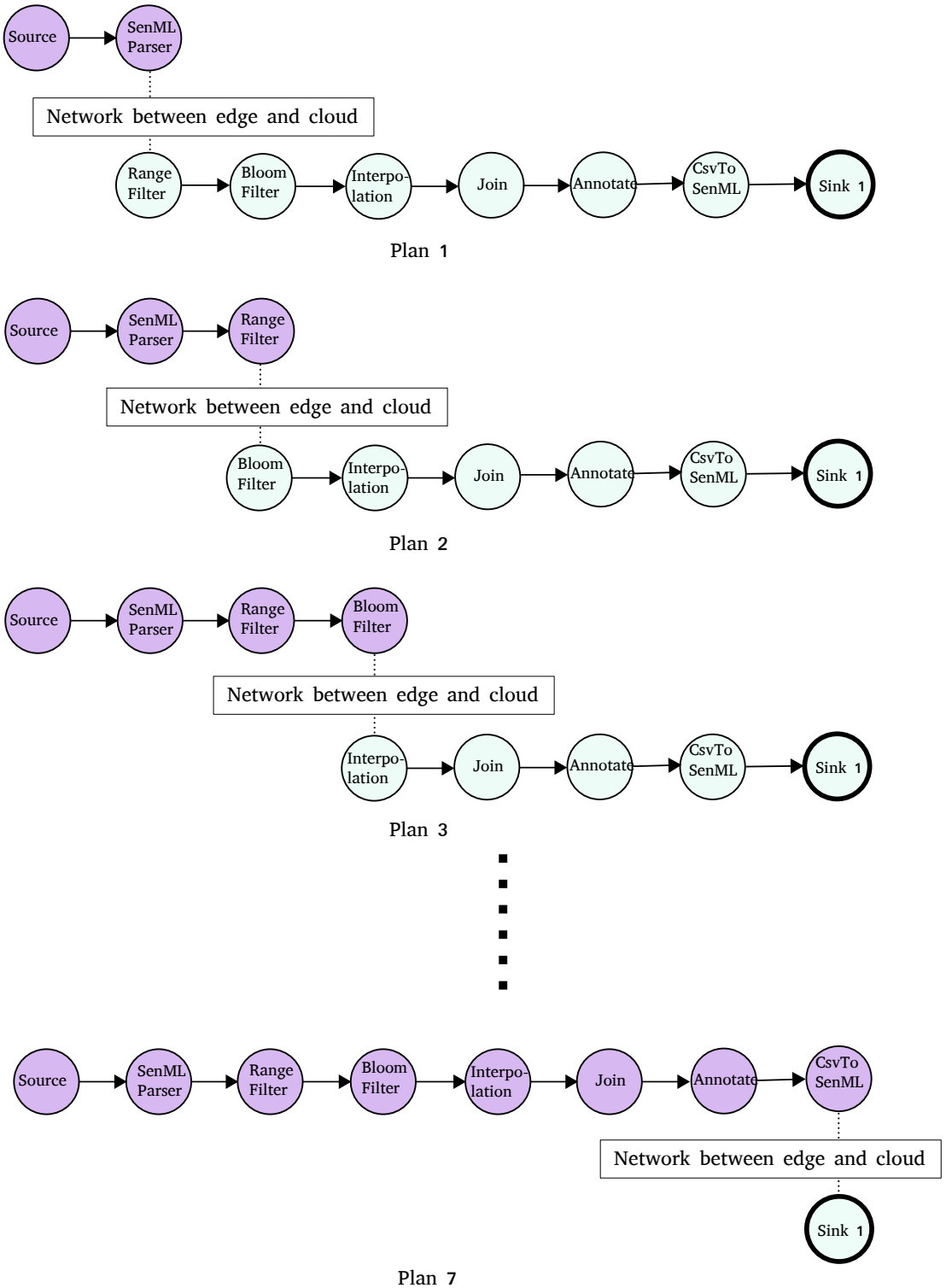


FIGURE 3.7: Possible placement plans for the simplified ETL application topology - purple nodes are placed on the edge while blue nodes are placed on the cloud.

for each DSP application executed on each host network, and can be used to monitor the performance of each application on either the simulation or the real experiment.

3.4.2.1 Delay Measurements

Each individual operator in the DSP application topology appends the processing time taken for a particular event and the sum of these individual processing times is recorded as the per event processing time for each event. When a network transmission occurs, the network transmission time for each particular event is recorded as well. The total latency per event is defined as the sum of individual processing latencies and network transmission latencies of the particular event. The same approach is used in both the simulation and the empirical set-up.

3.4.2.2 Power and Energy Measurements

To have a more clear estimate on the power consumed by edge equipment in the experiment set-up, we designed a power meter and a set of scripts to measure the power consumed by our Raspberry Pi device while the DSP application is being executed.

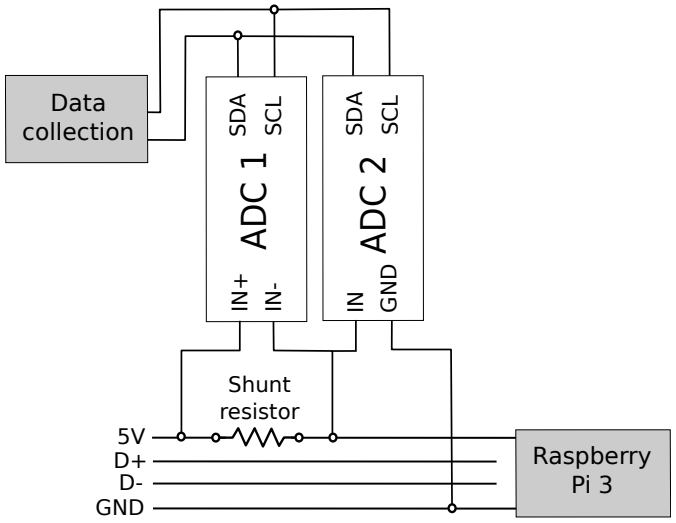


FIGURE 3.8: The power measurement set-up for the Raspberry Pi 3 devices. ADC 1 and 2 are used to measure the voltage across the shunt resistor and the input voltage respectively. Data collection scripts record the power consumption periodically, using the voltage and current measurements.

Raspberry Pi's are devices that can be powered via USB. The power consumption is hence measured by inserting a shunt resistor in the 5V USB line as depicted in Figure Figure 3.8. The resistor causes a voltage drop that is proportional to the current drawn by the Raspberry Pi. An Analog to Digital Converter (ADC) is used to measure the voltage across the resistor (*i.e.* ADC 1) in order to compute the amount of current drawn by the Pi, whereas ADC 2 measures the input voltage used to compute the power. ADC 1 and 2 are respectively an ADC Differential Pi and and ADC Pi Plus, both produced

by AB Electronics UK [8]. Both ADCs are based on Microchip MCP3422/3/4 that enable a resolution of up to 18 bits. Data collection scripts are executed by another Raspberry Pi device in the environment[9].

Prior to each configuration (*i.e.* placement plan) of the ETL or STATS application is executed on the experimental set-up (Figure 3.4), the data collection scripts are executed in order to measure the power consumption of the edge device periodically. Measured power consumption values are used to calculate the average power consumption by the edge device in each experiment afterwards.

In ECSNeT++, we use the power and energy model as described in Section 3.3.4 to record the total energy consumption for each execution of the different configurations of the application and calculate the average power consumption for the respective time period.

3.4.3 Calibrating the Simulator

In this section we describe how the simulator has been calibrated in order to accurately simulate the real world experiments for the ETL application.

The WLAN between the edge node and the access point has a limited bandwidth of 11Mbps and operates in IEEE 802.11b mode. The rest of the Full duplex Ethernet network operates at 1Gbps bandwidth. The host network configurations as described in Table 3.5 are mirrored in the host configurations of the simulator and the experimental set-up. For the power consumption measurements, we have acquired the values illustrated in Table 3.6 on the experimental set-up and we have configured ECSNeT++ using the same values. For all the other parameters, we use the default configuration values in OMNeT++.

First, each task in each application is profiled to measure the per event processing time (in nano second precision) for each CPU architecture. These measured values are used as the per event processing time, $\delta_{T_i V_j}$ value in each placement plan. Then the network transmission latency between edge and cloud is measured for each case and these measured latency distributions are used for the configuration of latencies in the simulator for each plan. While the event generation rate is varied, the initial event size of the source, and selectivity and productivity ratios of the operators are measured in the real application and the same values are used in the simulation set-up. Each application is executed until 10000 events are processed, and the experiments are concluded once

[8] https://www.abelectronics.co.uk
[9] Available at: https://github.com/assuncaomarcos/pi-powermeter

TABLE 3.5: Network configuration used in the experimental set-up and the simulation.

Property	Value
WLAN operation mode	IEEE 802.11b
WLAN Bit rate	11 Mbps
WLAN frequency	2.4 GHz
WLAN radio bandwidth	20 MHz
WLAN MTU	1500 B
WLAN transmission queue length	1000
WLAN RTS threshold	3000 B
WLAN MAC retry limit	7
WLAN Transmission power	31 dBm
Ethernet Bit rate	1 Gbps

TABLE 3.6: Measured average power consumption values used for configuring Raspberry Pi 3 Model B devices in the simulation. See Equation (3.1) for usage of these values.

Property	Value (W)
Φ_{idle}	1.354
$\Phi_{cpu}(100\%)$	0.258
$\Phi_{cpu}(400\%)$	0.955
$\Phi_{wifi,idle}$	0.118
$\Phi_{wifi,up}$	0.175
$\Phi_{wifi,dn}$	0.132

the measurements are acquired for each of these processed events in both the simulator and the experimental set-up.

3.4.4 Evaluation of Latencies

Figure 3.9 and Figure 3.10 show all the latency measurements for fixed source event generation rate of 10 events per second and 100 events per second respectively for all the considered placement plans of ETL application (Figure 3.7). Similarly, Figure 3.11, and Figure 3.12 show the latency measurements of all the placement plans of STATS app running at 5 and 10 events per second. Each experiment is concluded when 10000 events have been processed.

We conduct a qualitative analysis of the observed results for each scenario.

3.4.4.1 Network Latency

We first evaluate the network latency parameter for each scenario. In Figure 3.9 and Figure 3.10 the network latency values follow the same trend even though the real

experimental set-up shows higher interquartile range with respect to the simulation set-up, where the variations in the simulated values are limited. In the experiments, we calculate the network latency as the time taken by an event to leave an edge device and arrive at the cloud server, which includes all the latencies caused by the management of packet delivery over the communication channel such as CSMA/CA, rate control, packet retransmission and packet ordering caused by failures, etc., which results in the variations we observe in the results. It is not possible to simulate an identical environment since it is not possible to measure these intricacies. Since the IEEE 802.11 implementation of OMNeT++/INET has the same characteristics and due to the close calibration of the simulator (Section 3.4.3) we can observe some variation in the network latency measurements of the simulation specifically when the amount of packets transmitted is high. The lack of variation in the Plan 5, 6 and 7 when compared with the Plan 1, 2, 3, and 4, is caused by the lack of the amount of events transmitted by Plan 5, 6 and 7. The "`Join`" Task which is the terminal task placed at the edge device at the Plan 5, has a selectivity ratio of 0.2 (see Figure 3.5) which results in ∼5 times reduction in the amount of events transmitted from the edge to the cloud which is also affecting the Plans 6 and 7. However, the slightly higher variation in Plan 7 is due to the `CSV to SenML` task which is placed on the edge nodes in this plan. This task has a productivity ratio of 2 which results in larger events being transmitted from the edge to the cloud over the network which in turn causes the slight variation that is visible.

In Figure 3.11, and Figure 3.12, Plan 1, 2, 3, 4 and 5 show higher variation in network latency which is caused by the structure of the topology where a large number of events are transmitted between the edge and the cloud due to multiple dataflows in the topology between the `SenML Parser` and the downstream operators.

By observing the trend in these figures we can conclude that for each case in each placement plan, the simulation can be used as an approximate estimation for the network latency measurements in a real experiment with similar parameters.

3.4.4.2 Processing Latency

As illustrated in Figure 3.9, Figure 3.10, Figure 3.11, and Figure 3.12 the processing latency measurements of the simulation closely follow the same trend as the experiments. However, the processing latency in the experimental setup has higher variation which is expected due the complex processing architecture present in real devices which involves context switching, thread and process management, IO handling and other overheads. We measure the per event processing latency in each task in each placement plan and use the mean per event processing latency for each task in the simulator. Therefore the

lack of variation in the processing latency measurements is visible in the illustrations. In Figure 3.9, the higher variation in the Plan 4 is caused by the measured processing time difference between the two Raspberry Pi devices which is used as processing latency per each task in each device configurations.

The processing model of ECSNeT++ is not as intricate as a real CPU architecture. We believe that making the processing model complex (and realistic) may have a larger impact in some simulation scenarios which would generate more accurate processing latency results.

3.4.4.3 Total Latency

Total latency is the sum of the processing latency and the network latency in each event. This is the most important measurement in terms of the quality of service provided to the end users by each DSP application. As illustrated in Figure 3.9, Figure 3.10, Figure 3.11, and Figure 3.12, the simulated total latency measurements closely follow the same trend as in the real experiments specially since the latency measurements are dominated by the processing time. With these observations it is also possible to conclude that the near-optimal placement plan for each DSP application scenario is the same in the real experiments and in the simulated experiments. Therefore, with these observations we can show that ECSNeT++ can be used to estimate the near optimal placement plans for Edge-Cloud environments with proper calibration where a real test environment is not available.

3.4.5 Evaluation of Power Consumption

As observed in Figure 3.13, Figure 3.14, Figure 3.15, and Figure 3.16, the calculated average power consumption values leave more detailed modelling and fine tuning to be desired on ECSNeT++. When the two event rates of each application are compared, processing lower number of events per second consumes less average power than processing higher number of events per second, because it takes more time to process 10000 events when event rate is low which results in decreasing the energy consumption per unit time i.e. power consumption. The idle power consumption is uniform in all the simulations since we have configured the simulator with the idle power consumption of 1.354W (See Table 3.6).

In Figure 3.15, and Figure 3.16, Plan 6 shows a large difference between the experiments and the simulations. In Plan 6, Multi Line Plot task is moved to the edge devices where it is writing the generated plots to the disk. However, unfortunately we have not

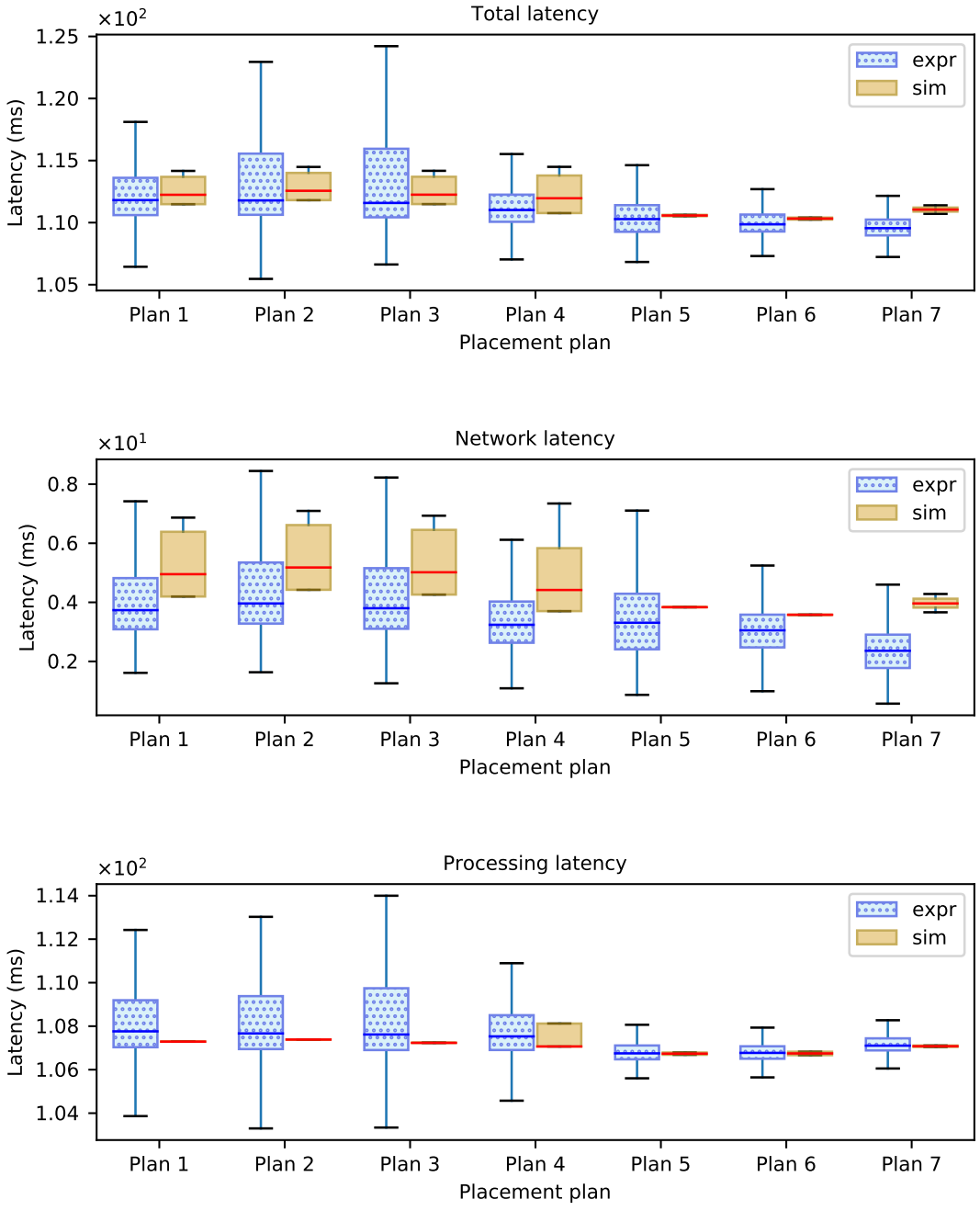


FIGURE 3.9: Comparison of experiments and simulation for the ETL application where the source is generating 10 events per second until 10000 events are processed. The total latency measurement is composed by the sum of network latency and the processing latency. Median latency of each plan on experiment and simulation are shown by the horizontal blue and red line respectively.

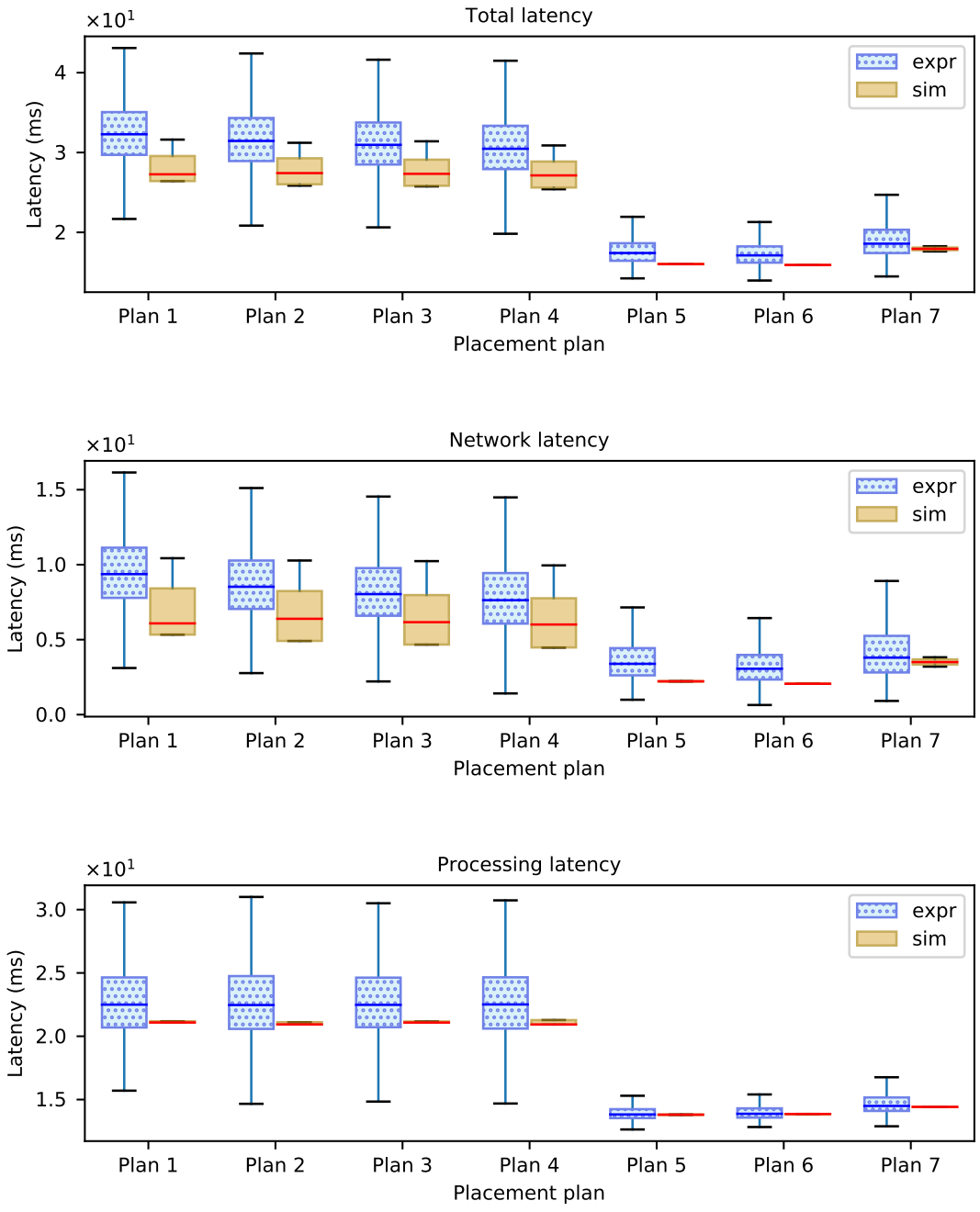


FIGURE 3.10: Comparison of experiments and simulation for the ETL application where the source is generating 100 events per second until 10000 events are processed. The total latency measurement is composed by the sum of network latency and the processing latency. Median latency of each plan on experiment and simulation are shown by the horizontal blue and red line respectively.

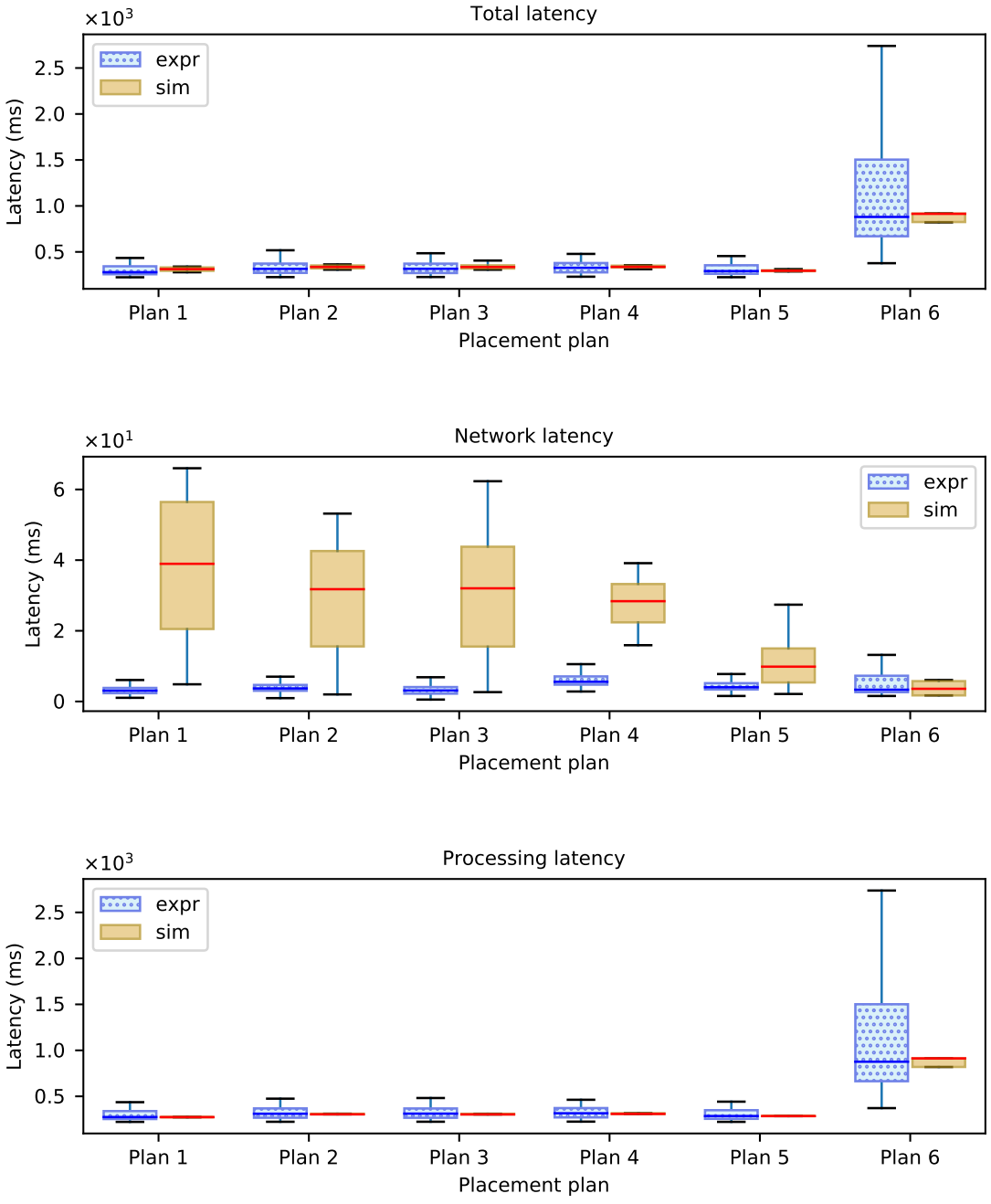


FIGURE 3.11: Comparison of experiments and simulation for the STATS application where the source is generating 5 events per second until 10000 events are processed. The total latency measurement is composed by the sum of network latency and the processing latency. Median latency of each plan on experiment and simulation are shown by the horizontal blue and red line respectively.

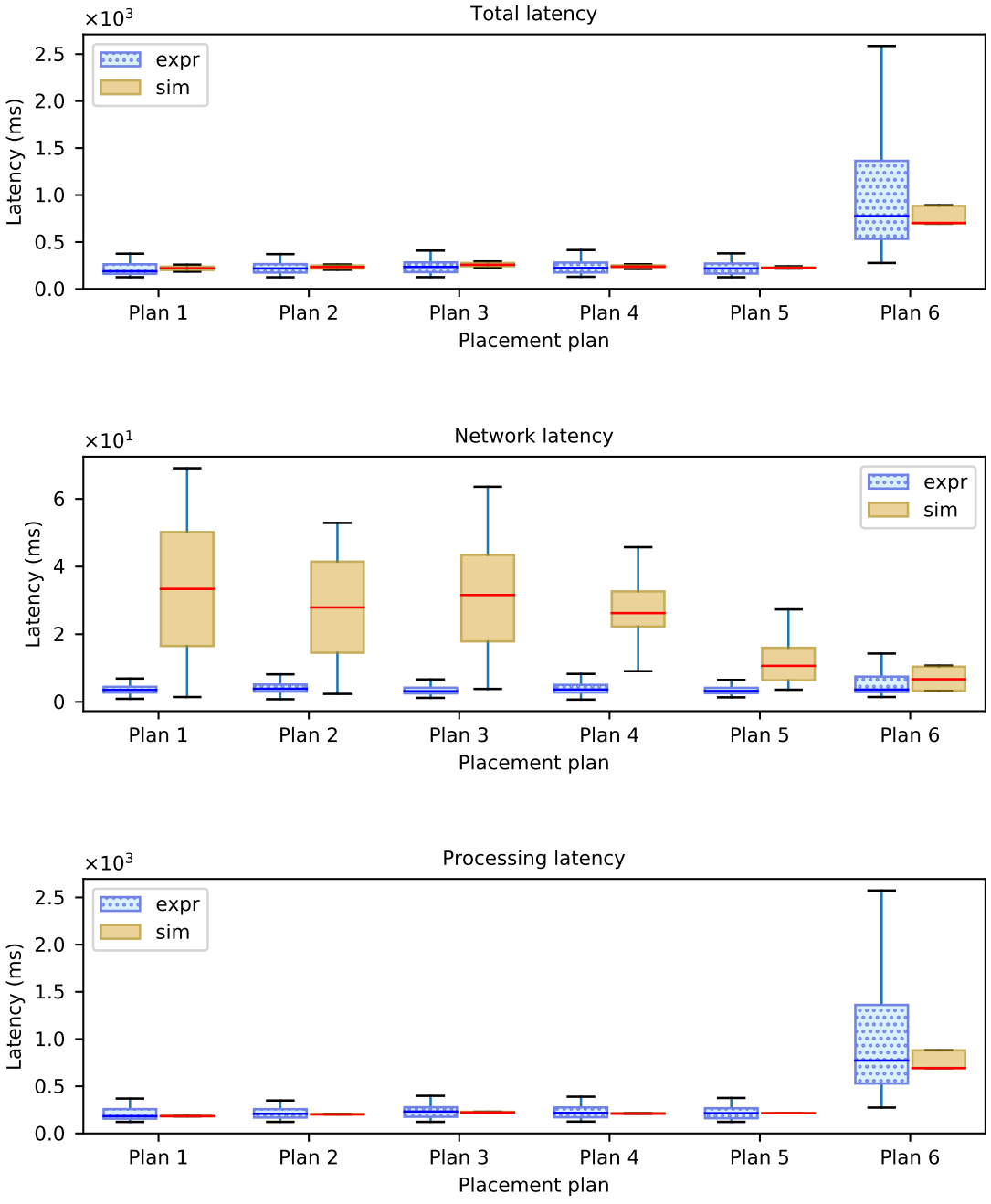


FIGURE 3.12: Comparison of experiments and simulation for the STATS application where the source is generating 10 events per second until 10000 events are processed. The total latency measurement is composed by the sum of network latency and the processing latency. Median latency of each plan on experiment and simulation are shown by the horizontal blue and red line respectively.

modelled that in our simulator's processing power consumption model which we believe is the reason for the observed significant difference in processing power consumption comparison.

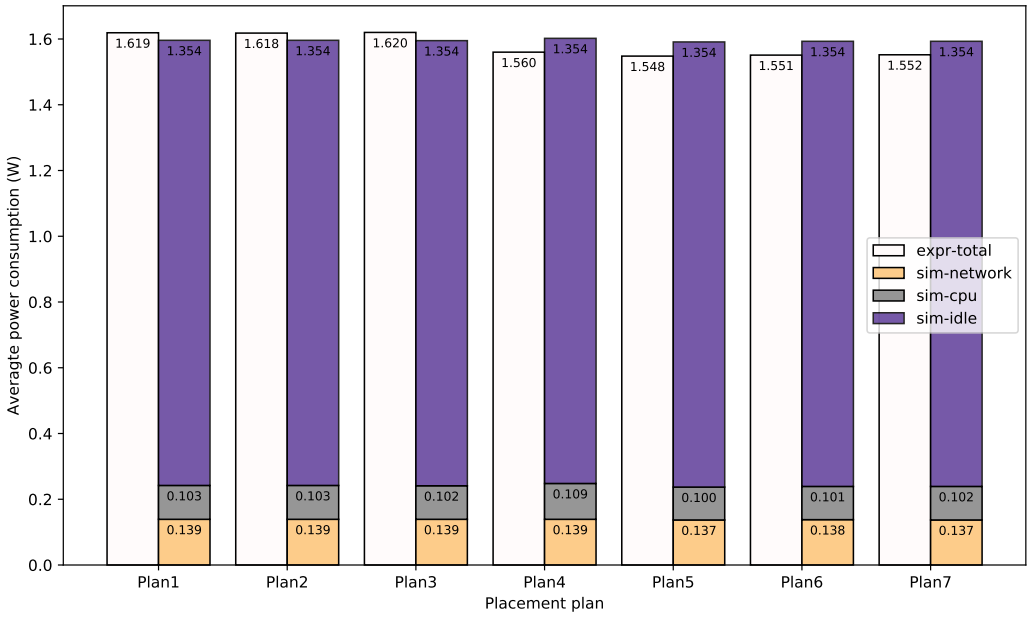


FIGURE 3.13: Comparison of average power consumption on the edge device in the experiments and simulation for the ETL application where the source is generating 10 events per second until 10000 events are processed. Each value in the plot shows the average power consumption of that component.

3.4.6 Extending the Network Model with LTE Connectivity

While previous experiments were conducted in an environment where edge devices were connected to the cloud through and IEEE 802.11 WLAN network, cellular networks are also being used widely for edge computing deployments [232][252]. Therefore, we have extended the network model of ECSNeT++ to use the SimuLTE[10] [252][256] simulation tool to introduce the LTE Radio Access Network (LTE RAN) connectivity. We use the LTE user plane simulation model of SimuLTE to develop a set of mobile devices, which we call `LTEEnabledPhone`, which acts as the User Equipments (UEs) that use LTE connectivity. In the simulation, these devices are then connected to the cloud through a base station known as eNodeB which allocates radio resources for each UE.

[10]http://simulte.com/

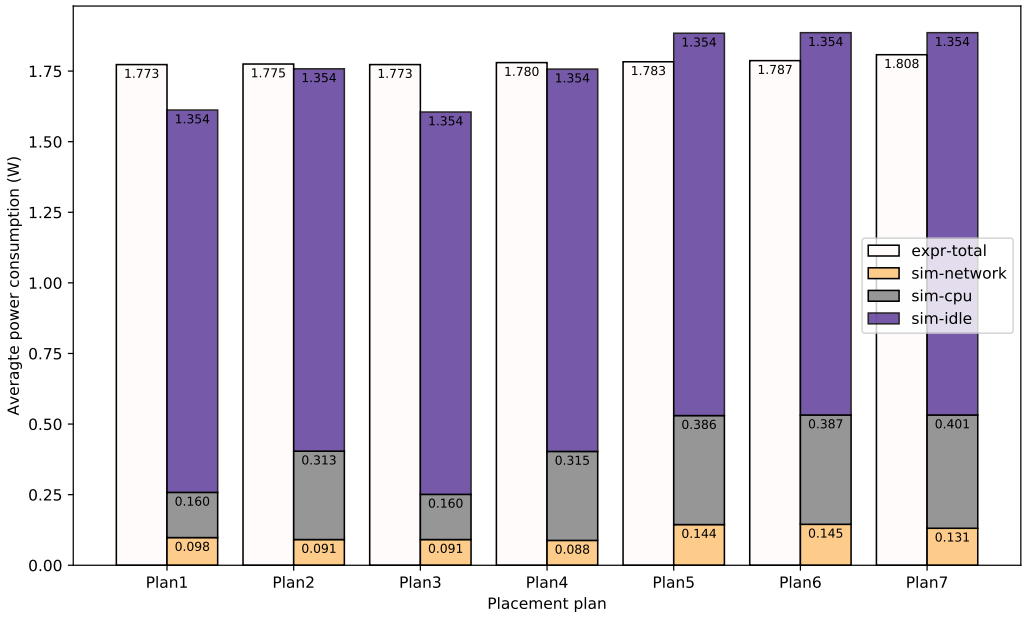


FIGURE 3.14: Comparison of average power consumption on the edge device in the experiments and simulation for the ETL application where the source is generating 100 events per second until 10000 events are processed. Each value in the plot shows the average power consumption of that component.

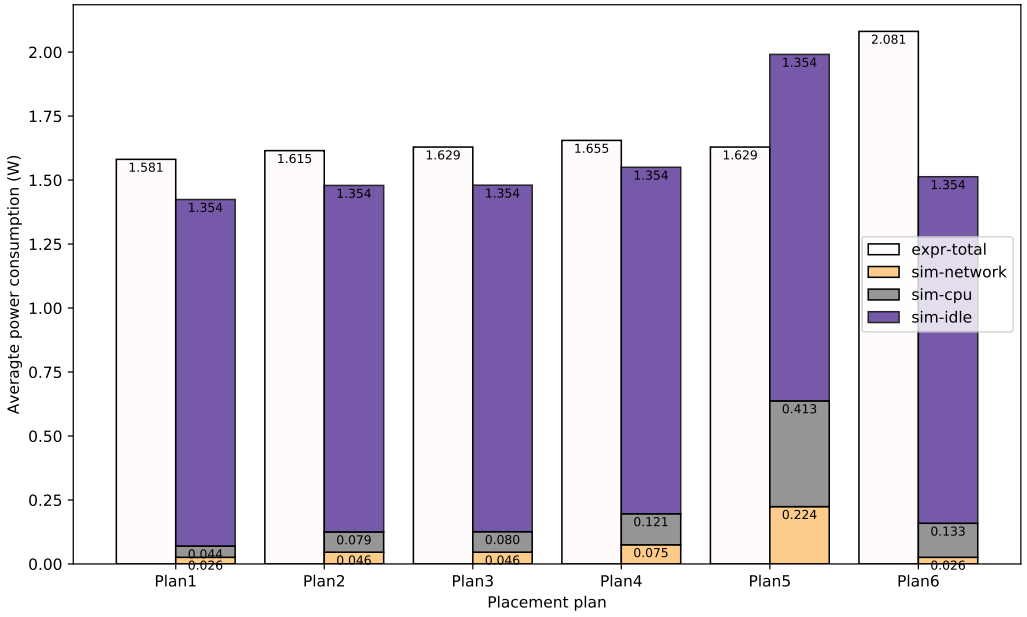


FIGURE 3.15: Comparison of average power consumption on the edge device in the experiments and simulation for the STATS application where the source is generating 5 events per second until 10000 events are processed. Each value in the plot shows the average power consumption of that component.

In addition, these UE devices move in a straight line, within a confined area of 90000 m^2,

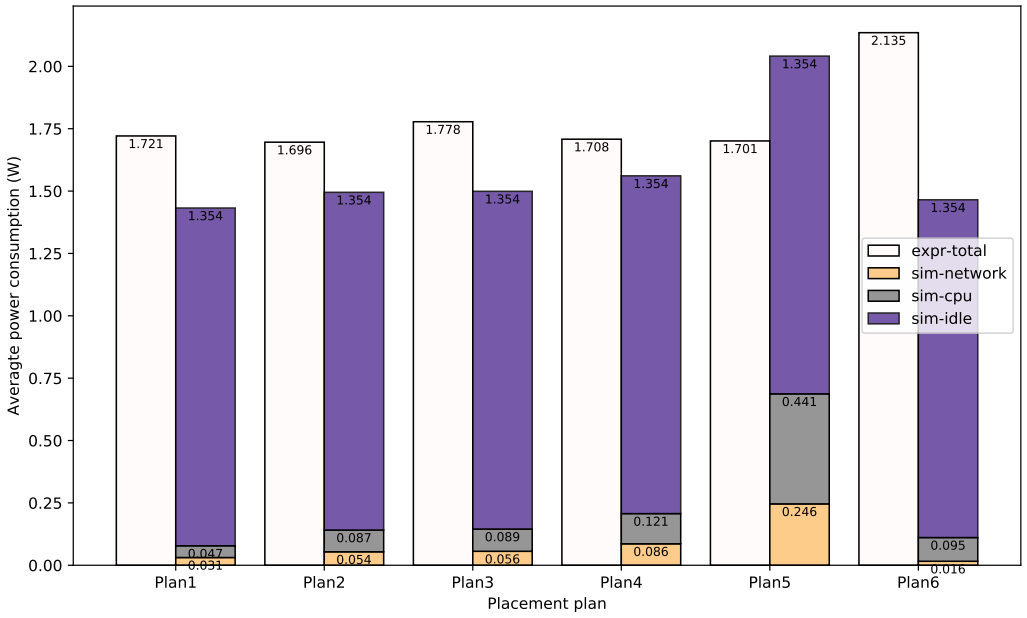


FIGURE 3.16: Comparison of average power consumption on the edge device in the experiments and simulation for the STATS application where the source is generating 10 events per second until 10000 events are processed. Each value in the plot shows the average power consumption of that component.

with a constant speed of 1 ms^{-1}. This is modelled as a linear mobility model (`LinearMobility`) set via the `mobilityType` configuration that is inherited from the INET framework[11].

The configuration parameters and the environment created for the LTE simulation are shown in Figure 3.17. The results of the simulation are illustrated in Figure 3.18. The simulation results illustrate that even with the speed and low latency of LTE connectivity, sharing the workload with edge devices is beneficial for stream processing applications in terms of lowering the total latency of events.

While we leave the comparison of these results against a real test environment for future work due to the lack of physical resources to set up such an environment, we believe these experiments show the extensibility of ECSNeT++ to use other networking models with minimum configuration.

3.5 Comparison with Related Work

ECSNeT++ belongs to the simulators category (Section 2.6.2) under the evaluation tools in the taxonomy presented in Chapter 2. As shown in Section 2.6.2, while there

[11] https://inet.omnetpp.org/docs/showcases/mobility/basic/doc/index.html

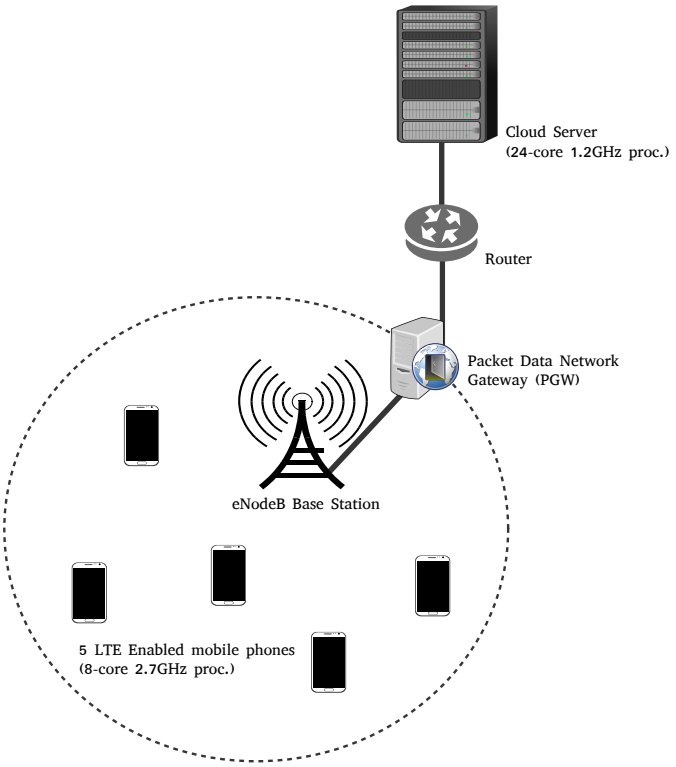


FIGURE 3.17: The simulated LTE environment where the edge devices (5 LTE enabled mobile phones) are connected to the cloud server using the LTE Radio Access Network.

are many tools to simulate cloud computing scenarios, only few simulation tools have been developed for edge computing scenarios. Here, we look at the existing literature on such simulation tools that are either developed for edge computing scenarios, or provided us a platform to build our work upon and compare them with ECSNeT++.

EdgeCloudSim [225] is a recent simulation toolkit built upon widely popular CloudSim tool [219]. EdgeCloudSim is built to simulate edge specific modelling, network specific modelling, and computational modelling. While EdgeCloudSim provides an excellent computational model and a multi-tier network model, the toolkit lacks the ability to simulate characteristics of a distributed streaming application without significant effort. Our work exclusively provides the ability to simulate characteristics specific to DSP applications which is crucial for testing optimal strategies for task placement on both edge and cloud devices. Sonmez et.al., also acknowledge the importance of accurate network simulations, and that network simulation tools such as OMNeT++ provide a detailed network model. However, instead the authors have implemented their tool on CloudSim due to the high amount of effort required to build the computational model on a network focused simulator such as OMNeT++. In our work we have implemented the computational model with significant effort while benefiting from the detailed network simulation aspects of OMNeT++.

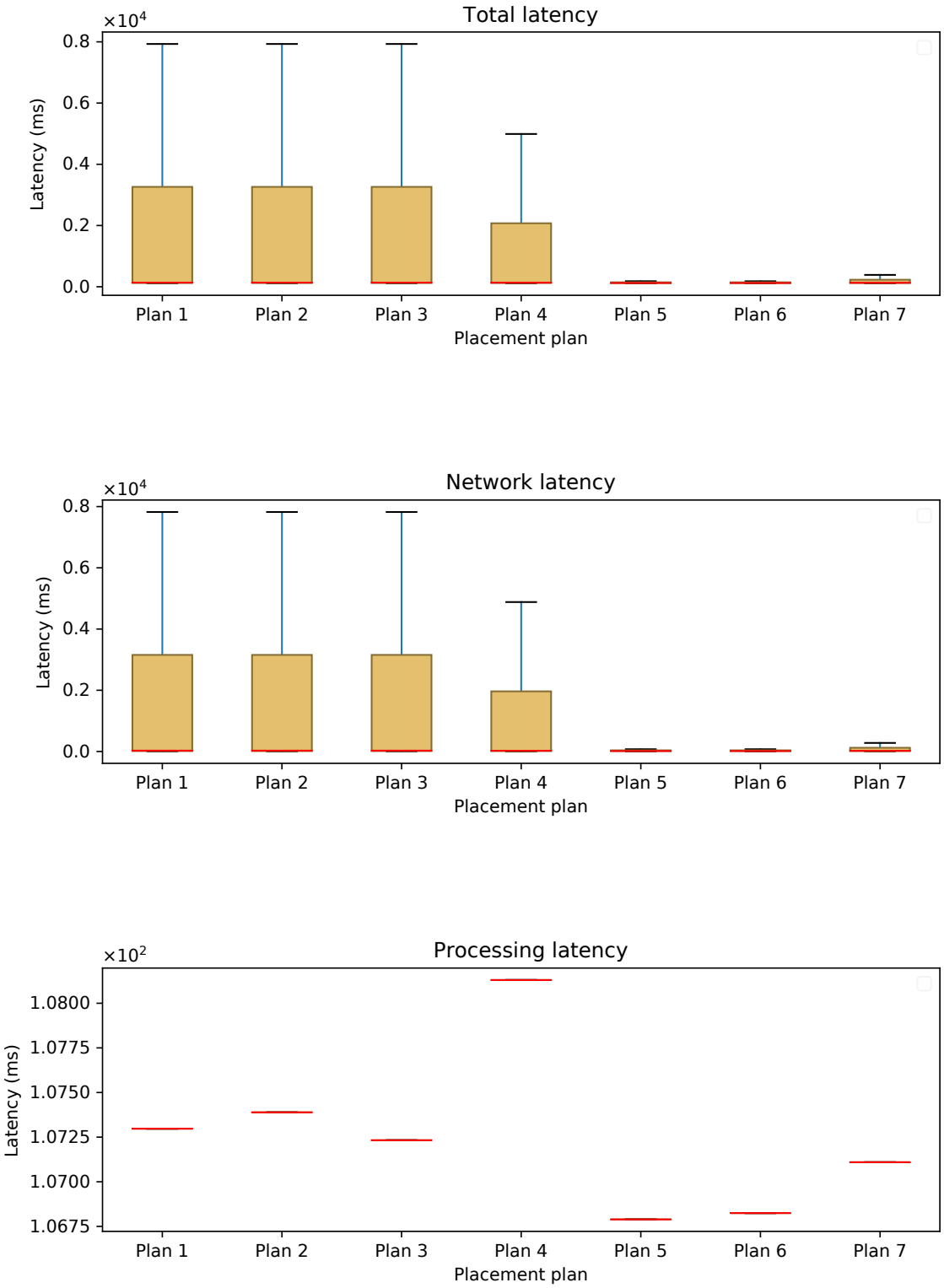


FIGURE 3.18: Simulation for the ETL application running on the mobile devices over an LTE network, where the source is generating 10 events per second until 10000 events are processed. The total latency measurement is composed by the sum of network latency and the processing latency. Median latency of each plan is shown by the horizontal red line.

Gupta et al. [92] introduce iFogSim, which is also built on top of CloudSim tool. iFogSim provides the ability to evaluate resource management policies while monitoring the impact on latency, energy consumption, network usage and monetary costs. Their applications are modelled based on Sense-Process-Actuate model, which represents many IoT application scenarios. However, when compared to iFogSim, our simulation tool provides a fine grained control on placement of tasks of DSP applications based on a placement plan where the applications are executed using our multi-core, multi-threaded processing model. It is important to note that unlike iFogSim, in our work we simulate a realistic IEEE 802.11 WLAN network along with an Ethernet network (with the ability to build more intricate networks with less effort), and uses UDP or TCP protocol for communication, which provides a significant similarity to a realistic application. The effects of these simulations are represented in our measurements.

CloudSim [219] provides a simulation tool for comprehensive cloud computing scenarios such as modelling of large scale cloud environments, service brokers, resource provisioning, federated cloud environments, and scaling. CloudNetSim [229] is a cloud computing simulation tool built upon OMNEST[12]/OMNeT++ toolkit with a state based CPU module with multiple scheduling policies where applications can be scheduled to be executed on these modules. We gained inspiration from the CPU module and the scheduling policies introduced in this work, when developing our computation model. Similarly, CloudNetSim++ [230], which focuses on the aspects of cloud computing, is also built on top of OMNeT++.

Table 3.1 compares our proposed solution against other relevant state-of-the-art modelling and simulation tools available in the literature. We have implemented our simulation toolkit to cover the limitations of the existing tools in assessing DSP applications running on edge-cloud environments. We have compared ECSNeT++ with a real experimental set-up to show the importance of calibration since none of the available simulation tools have conducted such a study. While there are few simulation tools for edge computing scenarios, they lack the ability to simulate the characteristics of a DSP application being executed on an edge-cloud environment, and lack a realistic network simulation which is critical. Our work is the first of its kind to implement these features on top of OMNeT++.

3.6 Conclusion

In this chapter we presented a unique simulation toolkit, ECSNeT++, which was built on a comprehensive model of data stream processing applications being executed on edge

[12]OMNEST is the commercial variant of OMNeT++ tool.

and cloud computing resources. ECSNeT++ extended the widely used OMNeT++ toolkit. We used multiple configurations of two realistic DSP application topologies, ETL and STATS applications, from the RIoTBench suite, to conduct the experiments. In order to verify ECSNeT++, we used a realistic experimental set-up with a Raspberry Pi 3 Model B device and a cloud node with a WLAN and LAN network where the same experiments with identical network, host device, and application configurations were executed on both the simulator and the real experimental set-up. We measured network latency, processing latency, the total latency per each event processed, and average power consumption by the edge device to show that when properly calibrated, the latency measurements of ECSNeT++ toolkit follow the identical trends as a similar real environment. In addition, we demonstrated the extensibility of ECSNeT++ by introducing LTE Radio Access Network connectivity for the network model of the simulations. We also believe that ECSNeT++ is the only verified simulation toolkit of its kind that was implemented with characteristics specific to executing DSP applications on Edge-Cloud environments. We believe that with the open source availability and extensibility of ECSNeT++, researchers will be able to develop effective and practical utilisation strategies for distributed data stream processing in edge-cloud infrastructure.

In our experiments, we did not focus on optimal task placements as we aimed at verifying the functionalities of the simulator. In order to utilise the edge cloud infrastructure we need to however devise optimal task placement strategies based on optimisation objectives. Therefore, in the following chapters we present mathematical models of stream processing along with static and dynamic task placement methodologies for utilising edge and cloud infrastructure for executing DSP applications. The ECSNeT++ simulator presented in this chapter is used to verify the suitability of these task placement methodologies.

Chapter 4

A Data Stream Processing Optimisation Framework for Static Task Placement

This chapter is derived from:

- **Gayashan Amarasinghe**, Marcos Dias de Assunção, Aaron Harwood, and Shanika Karunasekera, "A Data Stream Processing Optimisation Framework for Edge Computing Applications," *in 2018 IEEE 21st International Symposium on Real-Time Distributed Computing (ISORC)*, 2018, pp. 91–98.

4.1 Introduction

Ubiquitous computing has lead to the increasing availability of smart-sensors, wearable devices, smart-home devices, single-board computers, and other various connected smart-apparatus which produce billions of events that need to be analysed by some mechanism. Among the emerging application scenarios that exploit the analysis of such data, IoT can have a disruptive impact on society due to its pervasiveness and ability to connect a vast number of objects to the Internet. For instance, a utility company can monitor the power grid and water distribution network in order to identify possible problems with ageing infrastructure, such as leaks, predict demand, and plan accordingly [257].

Several of these emerging application scenarios produce streams of data whose processing is most valuable when carried out under short delays. One of the challenges consists

in provisioning the resources required to process these data streams. Cloud computing has been used to enable resource sharing, elasticity, data storage, and provide global presence under a pay-as-you-go business model. Although Cloud computing can provide the computing and storage resources required by these applications, emerging services demand data processing and analysis under very short response times. Edge computing [251] offers the ability to deploy services on resources located outside the cloud, that provide non-negligible computing power and are often located closer to where the data is generated. Although edge devices provide lower latency to data sources when compared to the cloud and their use is often free of charge, offloading data processing to the edges introduces challenges. Many edge devices are connected to the cloud via wireless networks and are powered by batteries.

We consider the deployment of data stream processing applications using cloud and edge resources where certain tasks are performed at the edge for reducing the end-to-end latency of processing events. Many DSP frameworks use a dataflow approach where incoming data traverses a directed graph of operators that execute transformations over the streaming data [43]. During application deployment, such operator tasks are placed onto cluster nodes to benefit from task and data parallelism. The placement of processing tasks across cloud and edge needs to consider computing, network and energy constraints. To achieve this, we model the scenario as a constraint satisfaction problem and employ a solver to acquire placement plans for cloud and edge devices.

More specifically, we make the following contributions: *(i)* we provide an optimisation framework that models the static task placement of data stream processing applications as a constraint satisfaction problem considering dataflow regions that can be deployed on edge or cloud computing devices, the computing requirements of tasks, and the power used for computing and communication; and *(ii)* we demonstrate the effectiveness of our framework through the use of the ECSNeT++ simulator presented in Chapter 3, that provides measurements of network latency and power consumption for a given data stream topology allocation and power consumption data. We used a number of realistic data stream topologies, specific to distributed stream processing on edge-cloud infrastructure.

The rest of this chapter is organised as follows. Section 4.2 describes the optimisation framework, whereas the modelling and simulation tool is described briefly in Section 4.3. The evaluation scenario and performance results are presented in Section 4.4. Section 4.5 discusses related work and Section 4.6 concludes the chapter.

4.2 Optimisation Framework

Due to the NP-Hard nature of the task placement problem [18][20][21], a near optimal task allocation strategy would try to map tasks in the DSP application to the nodes in the host network to improve the quality of service requirements, and reduce the resource consumption. However, the optimality of the task allocation relies on the accuracy of the optimisation framework and the imposed constraints. Therefore, we build an accurate and comprehensive framework that includes a system model and a constraint satisfaction model, to represent the problem using attributes that are critical for distributed stream processing on cloud and edge infrastructure. Before describing the constraints and the optimisation problem, we model the host network, and the DSP application first.

4.2.1 System Model

A host network or the infrastructure of a distributed stream processing system which consists of both cloud and edge resources, can be represented as a connected graph. Let this graph be $G_{res} = \{V^e_{res}, V^c_{res}, E_{res}\}$ where V^e_{res} represents the set of edge nodes, V^c_{res} represents the set of cloud nodes, and E_{res} represents the set of logical links between the nodes. Let $V = \{V^c_{res}, V^e_{res}\}$ such that $|V| = L$.

A distributed stream processing application, known as a topology, can be represented as a Directed Acyclic Graph (DAG) of source(s), operator(s), and sink(s). Let this graph be $G_{top} = \{So, Op, Si, E_{top}\}$ where So represents the set of sources, Op represents the set of operators, Si represents the set of sinks, and E_{top} represents the set of streaming data flow links between the vertices in the graph. So, Op, Si all represent the vertices in the DAG while E_{top} represents the edges of the DAG. Without loss of generality, we also declare the set of vertices in the topology *i.e.* tasks as T, such that $T = \{So \cup Op \cup Si\}$ and $|T| = K$.

There exists a subset of operators Op^e which are eligible to be edge executable (and optionally cloud executable) where $Op^e \subseteq Op$ and similarly a subset of cloud executable operators Op^c where $Op^c \subseteq Op$ and $Op = Op^c \cup Op^e$.

Next, we define a region, Reg of the distributed stream processing topology as the *closed neighbourhood*[1] of a given vertex (a source, an operator or a sink). For example, given any operator Op_i in the topology, a region that includes that operator is the closed neighbourhood of that operator, $Reg_{Op_i} = N_{G_{top}}[Op_i]$. With respect to the edge and cloud integrated system, we define an *Edge Executable region*, Reg^e as a region with

[1] A closed neighbourhood of a vertex v in a graph G, denoted by $N_G[v]$, is the induced subgraph of all vertices adjacent to v, including v itself.

no cloud executable tasks ($Op^c \cap Reg^e = \emptyset$), and a *Cloud Executable region*, Reg^c as a region with no edge executable tasks ($Op^e \cap Reg^c = \emptyset$) in the topology. Note that both Reg^e and Reg^c can contain sources and/or sinks. Next we define a *source neighbourhood region*, $Reg_{So} = \{Reg_{So_i} | So_i \in So \ \wedge Reg_{So_i} = N_{G_{top}}[So_i]\}$.

We define *Edge Executable, Source Neighbourhood regions (EESN regions)*, Reg^e_{So} as the set of regions with only edge executable operators in closed neighbourhood to a source such that any such region, $Reg^e_{So} = \{Reg^e_{So_i} |\ Reg^e_{So_i} = N_{G_{top}}[So_i] \wedge Op^c \cap Reg^e_{So_i} = \emptyset\}$. We then define k-EESN regions where $Reg^e_{So_i}(k)$ defines the k-neighbourhood closed graph of node So_i which includes all the nodes in the topology within a diameter k from the source So_i. Therefore the default EESN region defined earlier becomes a special case of k-EESN regions with $k = 1$ such that $Reg^e_{So_i} = Reg^e_{So_i}(1)$.

4.2.2 Characteristics of a DSP application

A DSP application consists of four components, source, operator, sink, and the dataflow. In a typical DSP scenario, the sinks, sources and operators, known as tasks, are placed and executed on a processing node. These different tasks in the topology have their own characteristics depending on the application. As described in detail in Section 3.2.2, each source has event generation rate and event size where as each operator has selectivity and productivity ratios associated with them. For this study, we assume these conditions remain as fixed values during the lifespan of the application.

Based on this system model, we move on to define the constraints that are relevant to the task placement problem. These constraints are used to limit the search space on potential placements and to acquire a near optimum placement to satisfy the requirements.

4.2.3 Constraint Satisfaction Problem

In R-Storm, Peng et al. [18] formulate a constraint satisfaction problem to model the different resource requirements of the tasks in the topology against the resource availability of a cloud based infrastructure. We use the same approach to model the task placement problem for a heterogeneous network with both cloud and edge nodes comprehensively as a part of our optimisation framework.

A Constraint Satisfaction Problem (CSP) can be defined as $\theta = (\alpha, \beta, \mathcal{C})$ where, $\alpha = \{\alpha_1, \alpha_2, ..., \alpha_n\}$ (a set of variables), $\beta = \{\beta_1, \beta_2, ..., \beta_n\}$ (a set of respective domains), and $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_k\}$ (a set of constraints).

A solution to a task placement problem, modelled as a complete and consistent CSP is a task allocation which can be defined as a set of mappings, $M : T \rightarrow V$ such that $T_i \mapsto M(T_i) = v_j$. An optimal solution to the CSP problem can be defined as: given an objective function F, where the optimal solution is achieved when F is minimised, find a mapping $M^* \in M$ such that $\forall M,\ F(M^*) \leq F(M)$.

Furthermore, we say two tasks in a distributed stream processing topology are *collocated* if the two tasks are adjacent in G_{top} and, after the allocation, the tasks are placed on the same node in G_{res}, *i.e.*, vertices $T_i, T_j \in T$ are collocated iff $T_iT_j \in E_{top}$ and $M(T_i) = M(T_j)$.

4.2.3.1 Variables and Domains

To translate the operator placement problem to a constraint satisfaction problem, we introduce K decision variables, $x_i \in X$, where $V^e_{res} = \{V^e_1, V^e_2, \dots V^e_{l_e}\}$, with $|V^e_{res}| = l_e$, and $V^c_{res} = \{V^c_{l_e+1}, V^c_{l_1+2}, \dots V^c_L\}$, with $|V^c_{res}| = L - l_e = l_c$ such that $\forall i \in \{1 \dots K\}$, $x_i = \{1 \dots L\} = \{1 \dots, l_e, l_e + 1, \dots L\}$. Based on that, we also derive $K \times L$ boolean variables $y_{ij} \in Y$ such that $\forall i \in \{1 \dots K\}$ and $\forall j \in \{1 \dots L\}$:

$$y_{ij} = \begin{cases} 1 \ ; \text{if the task } T_i \text{ is allocated to node } V_j \\ 0 \ ; \text{otherwise} \end{cases} \tag{4.1}$$

Therefore the variables and domains of the CSP can be translated in to the operator placement problem as $\alpha = \{X, Y\}$ and $\beta = \{\{1 \dots L\}, \{0, 1\}\}$.

We also define $K \times K$ boolean variables $z_{ij} \in Z$ to denote the adjacency matrix of the topology such that $\forall i, j \in \{1 \dots K\}$ where $T_i, T_j \in T$:

$$z_{ij} = \begin{cases} 1 \ ; \text{if the task } T_i \text{ is adjacent to task } T_j \\ 0 \ ; \text{otherwise} \end{cases} \tag{4.2}$$

Next we look at the relevant constraints of the problem. We introduce two types of constraints here, *Residency constraints* which are the constraints on the location of the tasks, and *Resource constraints* which are the constraints on the resource utilisation. We define these constraints using our system model.

4.2.3.2 Residency Constraints

First we look at the constraints on the placement of tasks based on the residency requirements. Due to specific hardware, software or other requirements such as privacy or policy requirements, some tasks may be required to be placed on specific resources. Let $v \subseteq V_{res}$ be the set of nodes, the tasks T_i are to be placed on. Then:

$$\forall T_i \in T : M(T_i) \in v \tag{4.3}$$

Since we are considering edge-cloud DSP applications, we can assume that the sources of the topology need to be placed on the edge devices. Therefore we can derive the following constraint from Equation (4.3), where $v = V^e_{res}$:

$$\forall T_i \in So : T_i : x_i \notin \{l_e + 1 \dots L\}, M(T_i) \in V^e_{res} \tag{4.4}$$

Similarly the cloud executable tasks in the topology need to be placed only on the cloud nodes. Therefore:

$$\forall T_i \in Reg^c : T_i : x_i \notin \{1 \dots l_e\}, M(T_i) \in V^c_{res} \tag{4.5}$$

Next we define the residency constraint on collocating the tasks on *k-EESN* regions with the respective source of the region. Here:

$$\forall T_j \in Reg^e_{So_i}(k) \wedge T_i \in So : M(T_j) = M(T_i) \tag{4.6}$$

Due to reasons similar to pinning tasks on specific resources, some tasks in the DSP topology would be restricted from being placed on certain nodes. We capture this requirement here. Let $\overline{v} \subseteq V_{res}$ be the set of restricted nodes per each task $T_i \in T$, then:

$$\forall T_i \in T : M(T_i) \in V_{res} - \overline{v} \tag{4.7}$$

4.2.3.3 Resource Constraints

When placing tasks on a cloud and edge integrated environment, the resource requirements become a critical component in the decision making process. We look at several resources in this regard, CPU, bandwidth, and energy. However, this model can be conveniently extended to other relevant and quantifiable resources as well.

Let $\zeta_{T_i V_j}$ be the number of CPU cycles needed to process an incoming event on task T_i for the specific hardware architecture of node V_j, and $\lambda^{in}_{T_i}$ be the ingress throughput (events/s) of task T_i. Let C_{V_j} be the clock speed of the CPU in Hz, $\eta_{V_j} \geq 1$ be the

number of cores on node V_j, ψ_{V_j} be the number of threads executed per core, and $w_C > 0$ denote the hardness of the CPU constraint. Then:

$$\forall V_j \in V_{res} : \frac{1}{\eta_{V_j}\psi_{V_j}} \sum_{\forall T_i} y_{ij}\zeta_{T_iV_j}\lambda^{in}_{T_i} \leq w_C C_{V_j} \tag{4.8}$$

Let $\beta^{out}_{T_i}$ be the size of an outgoing event (kB/event) at task T_i and $\lambda^{out}_{T_i}$ be the outgoing throughput (events/s) of task T_i. Let $B^{out}_{V_jV_k}$ be the bandwidth on edge $V_jV_k \in E_{res}$, and $w_B > 0$ be the hardness of bandwidth constraint. Then:

$$\begin{aligned} &\forall V_j, V_k \in V_{res}, \forall T_i \in T : \\ &\qquad y_{ij}\beta^{out}_{T_i}\lambda^{out}_{T_i} z_{i(i+1)} y_{(i+1)k} \leq w_B B^{out}_{V_jV_k} \end{aligned} \tag{4.9}$$

The energy consumption for an incoming event can be divided into three sections: (i) the energy consumed while receiving the event, (ii) the energy required to process the event, and (iii) the energy required to transmit the result to the next node. Let $\mu^r_{V_{k-1}V_j}$ (J/event) be the energy consumed when receiving an event at node V_j from node V_{k-1}, $\mu^p_{T_iV_j}$ (J/event) be the energy consumed for processing an event on task T_i on the node V_j, and $\mu^t_{V_jV_k}$ (J/event) be the energy consumed for transmitting an event from node V_j to node V_k. Let P_{V_j} (W) be the maximum power draw from the power source at node V_j, and let $w_P > 0$ be the hardness of power constraint. Then:

$$\begin{aligned} \forall V_j \in V_{res} : &\sum_{\forall V_{k-1}} \sum_{\forall T_{i-1}} y_{(i-1)(k-1)} z_{(i-1)i} y_{ij} \mu^r_{V_{k-1}V_j} \lambda^{in}_{T_i} \\ &+ \sum_{\forall V_k} \sum_{\forall T_i} y_{ij} z_{i(i+1)} y_{(i+1)k} \mu^t_{V_jV_k} \lambda^{out}_{T_i} \\ &+ \sum_{\forall T_i} y_{ij} \mu^p_{T_iV_j} \lambda^{in}_{T_i} \leq w_P P_{V_j} \end{aligned} \tag{4.10}$$

4.2.4 Optimisation Problem

The aim of our optimisation framework is to minimise the end-to-end latency of the DSP topology as it runs on the underlying network, since edge-cloud applications are known to suffer from high end-to-end latency [34]. Loosely, the end-to-end latency is the time between an input tuple being generated at a source node to when the results concerning that input tuple arrive at the sink node. We have found however that it is problematic to formulate this as an expression. We considered minimising the latency of the critical path in the topology, but doing so does not take into account the interactions between paths at operators, and such a minimisation function failed to achieve our aim. For example, an aggregation operator with multiple incoming edges may wait for the events from all the edges to produce a resulting event. Therefore as a heuristic function, we

considered *minimising the sum of individual latencies* over the topology and indeed in our experiments we have found that this objective function consistently gives the lowest end-to-end latency.

We breakdown the latency calculation into two parts, the computational latency and network transfer latency, and thereby calculate the sum of latencies:

$$L = \sum_{T_i \in T, V_j \in V_{res}} y_{ij} \delta_{T_i V_j} + \sum_{V_j, V_k \in V_{res}} \sum_{T_i, T_{i+1} \in T} y_{ij} z_{i(i+1)} y_{(i+1)k} \vartheta_{V_j V_k} \tag{4.11}$$

where $\delta_{T_i V_j}$ is the processing time of task T_i at the host V_j, and $\vartheta_{V_j V_k}$ is the latency of transferring an event between two hosts. Then our optimisation objective is to minimise L of G_{top} placed on G_{res}.

4.3 Simulation Design and Configuration

The following subsections briefly explain how the different elements of ECSNeT++ is used to design the experiments and then configured for evaluating our optimisation framework.

4.3.1 Host Network and Processing Nodes

We utilise the cloud and edge nodes available in ECSNeT++, arranged as shown in Figure 4.1. Here, the edge computing nodes are connected to the network via IEEE 802.11 WLANs using access points for each separate network. All the wired connections, used to connect the rest of the network to the cloud, are simulated as full duplex Ethernet links. We use the characteristics of Raspberry Pi Model B (PiB) devices and Raspberry Pi 3 Model B (Pi3B) devices as edge nodes, and we use the configurations of Intel Xeon Gold 6140 processor to model the processing capability of cloud nodes in ECSNeT++ toolkit. We run 2 sets of separate experiments with PiB and Pi3B devices as edge nodes (see Table 4.1).

All the communications in the host network are simulated using the UDP protocol. In addition to the lower layer capabilities, each device is configured with a UDP application layer to allow the DSP application components to communicate over the host network.

TABLE 4.1: Configurations for the edge and cloud devices.

	PiB	Pi3B	Cloud Node
Released Year	2012	2016	2017
Processor Freq.	700MHz	1.2GHz	2.3GHz
CPU Cores	1	4	18
RAM	512MB	1GB	16GB
Threads per core	1	1	1

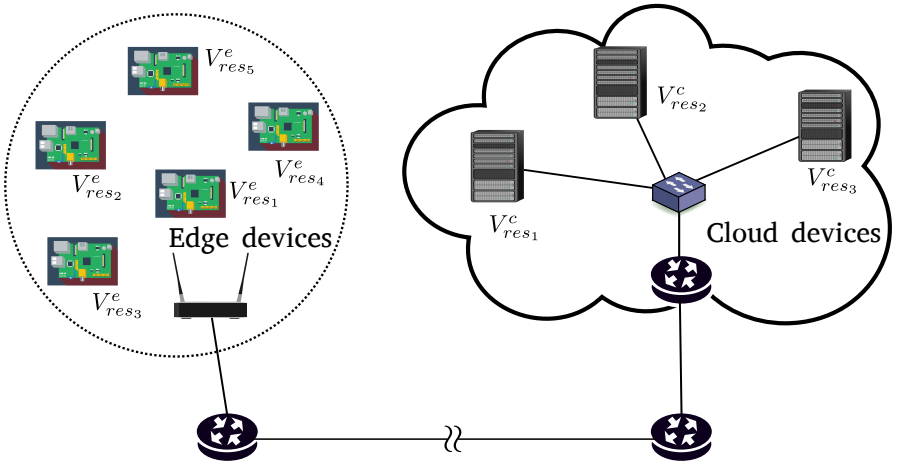
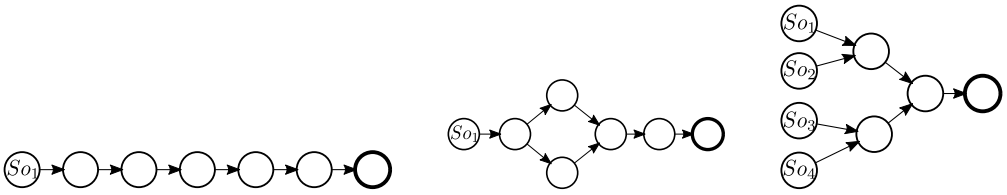


FIGURE 4.1: Host network with edge and cloud infrastructure.

4.3.2 DSP Applications

We employ two categories of DSP application topologies. The first category is a set of synthetic micro-benchmark topologies: linear, diamond, and tree as illustrated in Figure 4.2. A similar micro-benchmark based analysis approach is proposed by Peng et al. [18]. However, we have implemented our micro-benchmarks to represent common DSP application topologies with characteristics specific to edge-cloud integrated stream processing by adopting some of the topologies from other work [1][21][71]. We use these micro-benchmarks to establish a baseline for our optimisation framework. Here, the linear and diamond topologies have a single event source (see Figure 4.2(a), and Figure 4.2(b)) whereas the tree topology has four event sources generating events (see Figure 4.2(c)). As the second category we use a realistic edge-cloud application topology. We use the combined ETL+STATS topology from the RIoTBench suite [1] for this purpose.

Here, we assume the event generation rate is uniform and the initial event size is fixed and known. The set of topologies used for this simulation follow the same characteristics. We also assume the productivity ratio and selectivity ratio for each operator is fixed and known for the considered DSP topologies.



(a) Linear topology (b) Diamond topology (c) Tree topology

FIGURE 4.2: Micro-benchmark application topologies used for experiments.

Each DSP application used in the experiments is curated using the DSP task model of ECSNeT++ and configured using the characteristics of each application. A task can be placed on a processing node and can be connected according to the adjacency matrix of the DSP topology, to simulate the dataflow of a DSP application. Depending on the task placement policy, each source, operator and sink is placed on a processing node.

4.3.3 Processing Model

We have configured each host node with 3 parameters; η_{V_j}, C_{V_j}, and ψ_{V_j} from Table 4.1, that define how the multi-core, multi-threaded CPU model of ECSNeT++ behaves. In addition, we also use the Round Robin Scheduler for selecting the CPU core for processing each streaming event.

The measured per event processing time for each task of the DSP applications, is acquired by profiling each task on each host node that is simulated (similar to Section 3.3.3).

4.3.4 Power Model

As we are mainly interested in the power consumption of edge devices, we have limited the power consumption model to represent the power consumption of wireless communication between the edge devices and the access points, and the power consumed by their CPUs while processing data events at edge nodes.

The total power, Φ_{total} consumed at each edge device can be calculated by Equation (3.1). When configuring the power model of the ECSNeT++ simulator, we use the power models generated in the work of Kaup, Gottschling and Hausheer [255][253] as the power model of our Raspberry Pi devices as shown in Table 4.2.

TABLE 4.2: Power consumption configuration of Raspberry Pi devices.

Parameter	Power Consumption(mW)	
	PiB	Pi3B
Φ_{idle}	1577	1488
$\Phi_{wifi,idle}$	942	764.5
$\Phi_{wifi,dn}$	1020.61	822.1
$\Phi_{wifi,up}$	1112.38	704.431
$\Phi_{cpu}(u = 95\%)$	171.95	588.145
$\Phi_{cpu}(u = 5\%)$	9.05	30.955

4.4 Evaluation

Our experiments are based on using the placement plans generated by the optimisation framework introduced in Section 4.2. The plans are used as the input for the simulation model described in Section 4.3. We measure the end-to-end delay of processing an event and the energy consumption of the edge devices during the execution. The presented results are averages of five experimental runs varying the simulation seed with simulation configurations of Table 4.3.

TABLE 4.3: Parameters used for simulation.

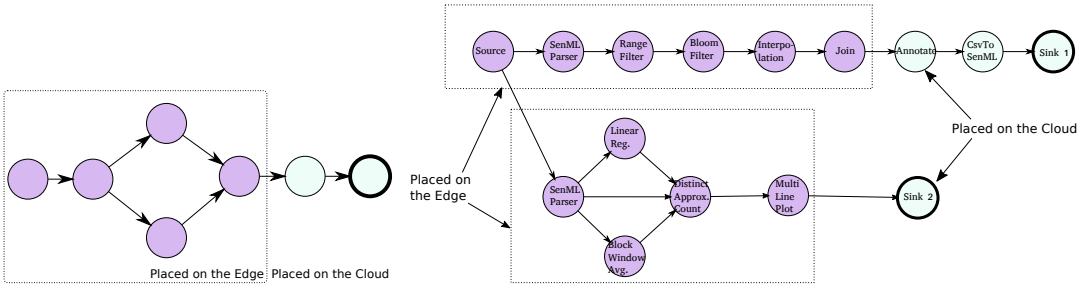
Parameter	Value
# Events	100000
# Edge Devices	50
# Cloud Devices	5
Warm-up Period	2s
WAN Bandwidth	6Mbps
LAN Bandwidth	100Gbps

4.4.1 Task Placement Plans

Task placement defines how each task of an application is deployed onto the available host nodes. Since we assume that the event generation rate and event size are fixed for each event source, and the selectivity and productivity ratios are fixed for each operator, only static task placements are considered during these simulations.

For each synthetic DSP topology, we execute three placement plans separately to test our hypothesis; *Cloud-Only (CO)* plan places all source nodes of the topology on the edge layer while the rest of the processing happens at the cloud layer, *Edge-Only (EO)* plan places everything except the sink nodes in the topology on the edge layer, the sink nodes are placed on the cloud layer. *Framework-Optimised (FO)* plan places the placement allocation generated by our optimisation framework on the topology, which includes a

shared placement of nodes in the topology between the edge and the cloud layer, e.g. as shown in Figure 4.3(a) and Figure 4.3(b). For the ETL+STATS application, we also execute each possible task placement plan separately (42 different placement plans including FO, CO and EO placement) and collect our latency and power consumption measurements to show that our approach can produce optimal results and compare the results against an exhaustive brute force approach.



(a) FO-placement of the synthetic diamond micro-benchmark topology

(b) FO-placement of ETL+STATS application topology

FIGURE 4.3: Topology and task placement of a selected synthetic micro-benchmark and the ETL+STATS application.

To obtain the FO placement plans for each topology, we implemented our constraint satisfaction and optimisation framework using the open-source MiniZinc constraint modelling language[2] [147][148]. Then we solve the model using the open-source Gecode solver [258] to acquire each FO placement plan.

4.4.2 Results

After executing the simulations for each micro-benchmark and the ETL+STATS application topology with multitude of placement plans, the collected end-to-end latency and energy consumption measurements are analysed.

4.4.2.1 Performance of the Micro-benchmarks

We first evaluate the effectiveness of the FO placement generated by the optimisation framework for the 3 synthetic micro-benchmark topologies in our simulation.

Figure 4.4 illustrates the end-to-end latency improvement of the micro-benchmark topologies. As observed here, for all 3 micro-benchmark topologies on both PiB and Pi3B devices, the Framework-Optimised (FO) placement results in $\approx 54\%$ reduction in mean end-to-end latency when compared with the Cloud-Only (CO) placement. When you

[2] https://www.minizinc.org/

compare the FO placement against the naive EO placement, in the Linear topology the reduction is ≈ 19%. In the Tree topology, the reduction is ≈ 8%, while in the Diamond topology the reduction is slightly more than ≈ 1%. These results show that offloading some tasks to the edge devices yields lower end-to-end latency depending on the application. However, a naive placement decision of placing all the tasks except the sinks on the edge devices does not guarantee better end-to-end latency since there are other factors that affect the placement decision, such as the computation, bandwidth and power consumption requirements of the tasks, as well as other constraints that affect their placement. Therefore, our experiments demonstrate that the FO placement plans generated by the optimisation framework, results in achieving better end-to-end latency results in these micro-benchmark topologies, where optimising the sum of individual latencies produce lower end-to-end latencies.

In Figure 4.4, we also observe that overall, the Pi3B devices produce lower latencies than the PiB devices. The reason for this observation is that the Pi3B devices have significantly more processing power when compared with PiB devices (1.2 GHz × 4 cores vs 700 Mhz × 1 core). Hence the overall processing latency is lesser in the Pi3B devices which in turn results in producing less overall mean end-to-end latency than the PiB devices. Here, all the other parameters such as the network topology, network bandwidth, and the cloud nodes are identical in both edge device scenarios.

Table 4.4 shows the per-event mean energy consumption of the 3 micro-benchmarks on PiB and Pi3B devices with each placement plan. The total mean per-event energy consumption is the sum of the processing energy consumption and the network energy consumption. Overall, the processing energy consumption is highest in the EO placement which is followed by the FO placement and then the CO placement. In the EO placement, all the tasks except the sinks are placed on the edge, hence the highest amount of processing is done at the edge which results in the higher processing energy consumption. Similarly, the FO placement uses the next highest edge processing energy where the CO placement uses the lowest amount of processing since only the sources are located on the edge.

However, the network energy consumption cannot be compared trivially since the number of events emitted by the last task on the edge device and the size of the emitted events dictate the amount of energy consumed for transmitting these events, in addition to the other network communication overheads.

When the total energy consumption of the 3 micro-benchmark topologies are compared we can observe that the FO placement results in lower energy consumption than the CO placement. However, when compared with the naive EO placement, the FO placement performs better in all scenarios except for the Diamond and Tree topology on the PiB

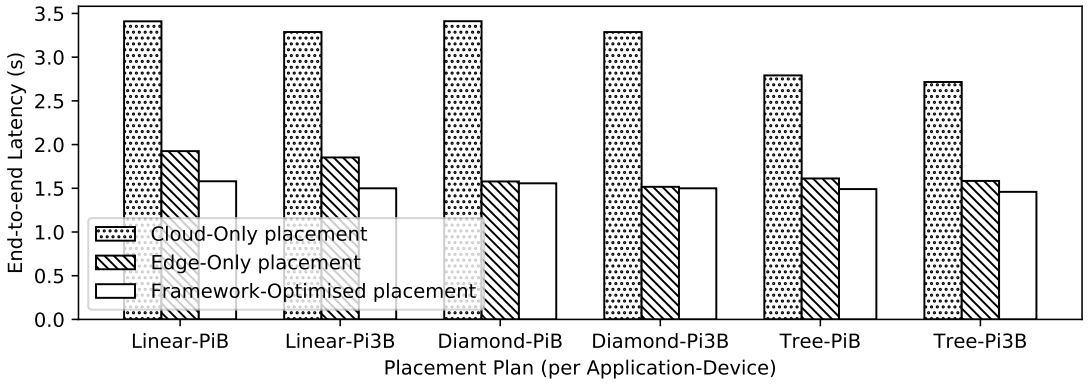


FIGURE 4.4: Mean end-to-end latency of micro-benchmarks.

devices, and the Diamond topology on the Pi3B devices. Even though the processing energy consumption is less in the FO placement here, if the last task on the edge device is transmitting more events to the cloud, the network energy consumption is higher on these devices. It is important to note that this does not violate our optimisation goal since still the end-to-end latency is lowest in the FO placement while the power constraint is satisfied.

4.4.2.2 Performance of the ETL+STATS application

We use the ETL+STATS application topology from the RIoTBench suite [1] as our application scenario. In order to simulate the event stream, we use the characteristics of the FIT dataset with a peak event rate of 500 events/s and source event size of 1024 bytes [117]. As shown in Figure 4.3(b), ETL+STATS application topology consists of 19 tasks in total, with 1 source, 12 operators and 2 sinks. In order to compare the performance of FO task placement, we exhaustively execute every possible placement plan for the ETL+STATS application individually and acquire the end-to-end latency of each plan.

Figure 4.5 illustrates that the FO placement plan results in 8% mean end-to-end latency reduction against the CO placement and 16% mean end-to-end latency reduction against the EO placement on PiB devices. Similarly, Figure 4.6 illustrates 14% mean end-to-end latency reduction against both CO and EO placements for the application running on Pi3B devices. Here, we have exhaustively run all the possible placement plans for the ETL+STATS application on the edge devices and show that FO placement produce the placement plan that results in optimal latency for these scenarios. Similar to the synthetic applications, the overall end-to-end latency measurements of the Pi3B devices is lower than the PiB devices due to the difference in the processing capabilities of these devices.

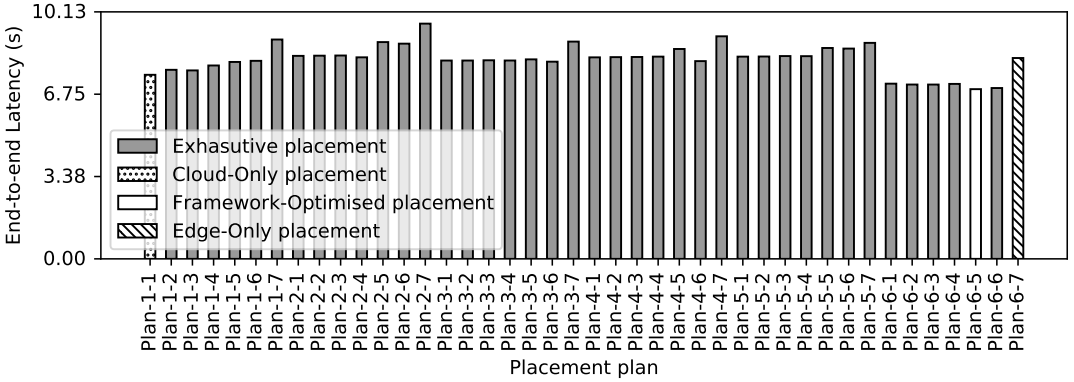


FIGURE 4.5: Mean end-to-end latency of ETL+STATS topology on PiB devices.

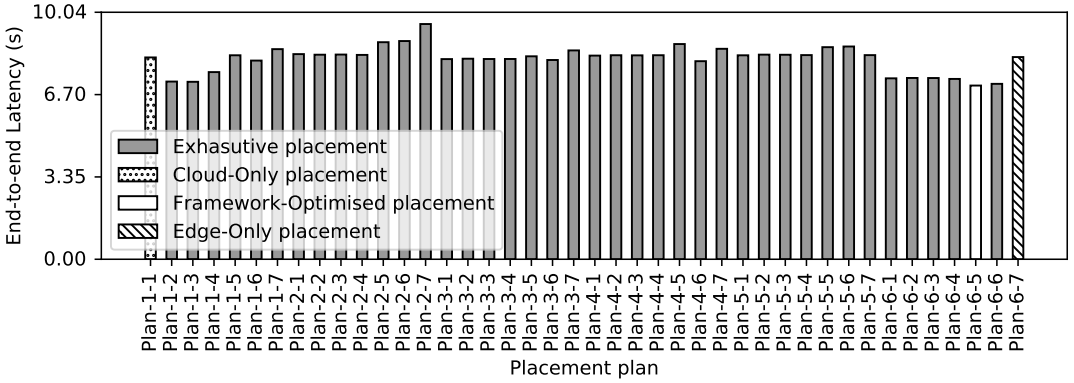


FIGURE 4.6: Mean end-to-end latency of ETL+STATS topology on Pi3B devices.

As seen in Table 4.4, $\approx 1\%$ reduction is observed in per-event mean energy consumption of the FO plan with the ETL+STATS application when compared with the EO plan on PiB devices and $\approx 5\%$ reduction on the Pi3B devices. However, $\approx 14-15\%$ improvement of per-event mean energy consumption is achieved when compared with the conventional CO approach on both device types. The high energy consumption with the CO plan is caused by the two SenML Parsers which are the last operators located on the edge device in the CO placement. These two operators each has a selectivity of 5 which produces 5 events for each incoming event that needs to be transmitted to the cloud, causing higher networking energy consumption. Each CO, EO and FO placement plan consumes processing energy comparable to the number of tasks placed on the edge devices in each plan, e.g. the EO placement consumes higher processing power due to the placement of every task except for the sinks on the edge devices when compared to the CO and FO placements. Overall, the PiB devices demonstrate lesser energy consumption for processing and higher energy consumption for networking in contrast to the Pi3B devices, representing the energy signature of each device as configured.

TABLE 4.4: Per-event mean energy consumption of micro-benchmark topologies (Values in bold show the minimum energy consumption of each topology on each edge device.).

Edge Device	Topology	Plan	Energy Consumption (mJ/event)		
			Processing	Network	Total
PiB	Linear	CO	4.7572	18.0047	22.7619
		EO	4.7577	17.3956	22.1533
		FO	4.7574	16.6788	**21.4362**
	Diamond	CO	4.7572	18.0047	22.7619
		EO	4.7577	16.715	**21.4727**
		FO	4.7575	17.0586	21.8161
	Tree	CO	2.3787	17.3625	19.7412
		EO	2.3790	16.3860	**18.7650**
		FO	2.3789	16.4862	18.8651
	Application	CO	2.288	31.423	33.711
		EO	3.714	25.086	28.8
		FO	2.992	25.59	**28.582**
Pi3B	Linear	CO	4.5560	16.8250	21.3810
		EO	4.5569	16.2078	20.7647
		FO	4.5563	15.7779	**20.3342**
	Diamond	CO	4.5560	16.8250	21.3810
		EO	4.5569	15.4935	**20.0504**
		FO	4.5565	15.6222	20.1787
	Tree	CO	2.2781	16.2956	18.5737
		EO	2.2787	15.5276	17.8063
		FO	2.2785	15.3055	**17.5840**
	Application	CO	9.255	27.055	36.31
		EO	12.667	20.106	32.774
		FO	10.321	20.837	**31.157**

4.5 Comparison with Related Work

This section compares the prior studies against our study from the perspective of *(i)* task placement strategies and *(ii)* distributed data stream processing in heterogeneous environments (*e.g.* comprising edge and cloud resources). Based on the taxonomy presented in Chapter 2 this study can be categorised as a static task placement framework that models the problem as a constraint satisfaction problem with a mono-objective optimisation on time.

4.5.1 Task Placement Strategies

Task placement problem with several solutions based on different application scenarios, frameworks, topology structures, underlying infrastructure and expected outcomes are present in the state-of-the-art [15][17][16].

R-Storm [18] provides a resource-aware scheduler for Apache Storm based on task requirements and the availability of three resource types – CPU, memory, and bandwidth. It proposes a formulation that corresponds to a Quadratic Multiple 3-Dimensional Knapsack problem and offers an approximation algorithm for calculating the operator placement by finding the distance between the resource demands and resource availability in a 3D-resource space. Similar to R-Storm, we model the placement scenario as a constraint satisfaction problem, but our model focuses on environments comprising edge and cloud resources.

Ghosh and Simmhan [19] present a placement strategy for complex event processing that considers computing and network latencies, throughput, and power usage at each cluster node. The authors use genetic algorithms (GA) to acquire a fast approximation to the problem and compare it against a brute force (BF) method that becomes intractable with large problem sizes. Their analysis focuses on the efficiency of the GA approach.

Cardellini et al. [20][21] focus on modelling the operator placement problem by considering the availability of nodes, cost of operator execution on nodes, response time, and network QoS metrics. Their comprehensive series of studies is designed for heterogeneous resources as well. However, they do not consider the residency constraints along with the resource constraints which is also critical for the task placement problem. It is important to note that the authors consider the task replication and parallelism which we do not consider. Instead, we only implicitly consider task replication with the available multiple edge devices where we allow to place replicas of the same task onto more than one device in a region, and assume that only one instance of a task is expected to be run on a given edge device. However, it is possible to consider parallelisation of tasks in our model by modifying the topology to include the number of parallel instances of each task.

In the works of [71], the authors handle operator scheduling on heterogeneous resources. Focusing on incorporating Graphic Processing Units (GPUs), they consider that scheduling decisions are aware of the performance characteristics of individual cluster nodes, of the incoming data, and the topology.

We have identified that these studies do not either consider the heterogeneity introduced by an edge-cloud environment and the challenges they impose on DSP applications, or have not conducted detailed experiments on edge-cloud environments. We comprehensively model an edge-cloud environment and DSP applications using a constraint based model that considers both resource and residency requirements, and evaluate the performance on an edge and cloud integrated network using appropriate micro-benchmarks and real world applications. We analyse the impact these constraints have on static

placement decisions in terms of end-to-end latency of processing events and energy consumption.

4.5.2 Distributed Stream Processing on the Edge

RIoTBench [1] present a set of benchmark DSP topologies for real-time IoT applications which we utilise for our experiments throughout this thesis. Mobistreams [216] focuses on using mobile devices for stream processing to reduce the overhead on the cellular network. The mobile network is divided into regions to which parts of a dataflow are assigned. The work focuses on checkpointing and reliability mechanisms. Using mobile devices for stream processing, Morales, Rosas and Hidalgo [232] also focus on improving fault tolerance mechanisms.

In addition, Beck et al. [36] propose a taxonomy of Mobile Edge Computing. They classify potential mobile edge computing applications based on four metrics of feasibility: power consumption, latency, bandwidth utilization, and scalability. They also identify that the main advantages gained through mobile edge computing are low latency results, offload-ability of workloads, and distribution of processing tasks. Even though the authors do not specifically consider any streaming applications, we believe the same conclusions can be applied in stream processing applications as well. However, the authors limit the concept of membership of Mobile Edge Computing only to the base stations of cellular networks. In our work, any device with the capability to execute an application, and logically situated below the cloud, can be a member of the edge and can be considered for task placement without any changes to the model.

4.6 Conclusion

We introduced a comprehensive optimisation framework that consists of a system model and a constraint satisfaction formulation, to represent the characteristics of DSP applications executed on edge-cloud environments where we considered the characteristics of the host network and the DSP application topology in our system model. Based on this system model, we introduced a set of resource constraints and residency constraints to represent the requirements of executing an application on an edge-cloud environment. By using our optimisation framework to optimise for the sum of individual latencies, and the ECSNeT++ simulation toolkit to simulate our edge-cloud DSP application scenario with a realistic host network, we showed that we could reduce end-to-end latencies of DSP applications by offloading some tasks to the edge nodes which in turn reduced the per-event energy consumption at the edge.

We showed that the generated Framework-Optimised placement plans for the simple micro-benchmark topologies provide slight improvements against a naive Edge-Only placement. However, in a complex realistic application topology, the FO plan resulted in $8-14\%$ reduction of mean end-to-end latency when compared with a conventional cloud-only placement, and $14-16\%$ reduction when compared with an edge-only placement. The FO plan also demonstrates $\approx 1-5\%$ improvement in terms of total per-event energy consumption against the edge-only placement, which is substantial in a DSP application that processes an unbounded event stream while also being able to improve the per-event energy consumption by $\approx 14-15\%$ when compared with the conventional cloud-only placement.

In this study we considered the application and environment conditions to remain static and as a result the produced placement plans were static. While initial static task placement was important, environment and load conditions could change during the application execution and as a result a static task placement may not be ideal. For such applications dynamic task placement may be desirable. However, frequent changes in task placement are detrimental to real-time DSP applications since continuous changes affect the end-to-end latency of producing results due to auxiliary work conducted for facilitating task placement changes. In the following chapter, we provide a solution that addresses this problem by devising a hybrid semi-dynamic task placement solution that utilises the static task placement solutions produced in this study in a Markov Decision Process.

Chapter 5

A Semi-Dynamic Task Placement Strategy for Data Stream Processing

This chapter is derived from:

- **Gayashan Amarasinghe**, Marcos Dias de Assunção, Aaron Harwood and Shanika Karunasekera, "A Two-Phase Semi-Dynamic Task Placement Strategy for Data Stream Processing on Edge Computing Infrastructure," *IEEE Transactions on Parallel and Distributed Systems*. (Under review)

5.1 Introduction

Numerous smart devices are becoming increasingly ubiquitous and essential to many services used by our society. Cisco predicts that by 2023 the number of Internet-connected devices will be three times the global population, and that over 70% of the global population will have mobile connectivity [259]. The rapid addition of such a vast number of connected devices will produce large amounts of data, much of it in the form of unbounded data streams that require timely processing in order to enable decision making on controlling the devices themselves or the environment in which they are deployed.

As depicted in prior chapters, data streams produced by connected devices are often processed using Data Stream Processing (DSP) techniques and frameworks such as Apache Storm or Apache Flink, which are generally deployed on the cloud to take advantage of

its scalability and pay as you go business model [43]. Edge computing has emerged as a paradigm to mitigate the issues of cloud only processing [34], [36], and [39]. Edge computing can offload some of the workload from the cloud by using devices located closer to the edges of the network for processing event streams that are generally created by the devices themselves – an approach that has proven to be useful in applications where the devices have adequate processing capabilities. However, as shown in Chapter 4, given the numerous parameters that affect a DSP application such as the event generation rate, event size, and the current load on the overall system, deciding how much workload to offload from the cloud and to which devices, is a complex task.

In many real world scenarios, parameters such as the event generation rate and event size vary depending on the time of the day and application type. Hence, a static task placement defined before the application starts and remains fixed is not optimal as placement decisions need to be taken continuously during application runtime considering its current state and load conditions. While a fully dynamic solution that makes task placement decisions by considering the on-going application status is desirable, it poses a great deal of overhead. In addition to the demand caused by monitoring the application for changes, the task placement decisions and their transitions require investigating a large number of resource and application configurations, which needs extensive computation time. When DSP applications require near-real-time results, this overhead can be detrimental to the quality of service delivered to end users.

To overcome this overhead and reduce the time to compute a task placement decision, we propose Two-phase Semi-dynamic Task Placement, an approach termed as TSTP that models the problem as an *Infinite Horizon Discrete Time Markov Decision Process*. In contrast to the existing literature where the placement decision (and task migration) is taken dynamically based on the current state of the application [160], [167], [159], and [163], TSTP selects a pre-decided placement plan. We also propose multiple heuristics to further reduce the state space of the problem to acquire fast placement decisions during the phases.

The first phase of our approach is a static system analysis to acquire the set of optimal placement plans with respect to the possible system configurations and environment states. The application in question is statistically analysed in order to model the distribution of event generation rate and generated event size supported by the application behaviour. Then based on a discrete distribution of these parameters, the latency optimal placement plans are acquired using a constraint programming technique introduced in our previous work on Chapter 4, such that a list of placement plans with respect to each parameter combination is acquired. The second phase is a dynamic decision making process based on the top placement plans found in the first phase. The activation

decision is modelled as a Markov Decision Process (MDP) which is solved to obtain a stationary policy. The state space of the MDP comprises combinations of placement plans acquired in the first phase and the environment condition in which the plan is optimal. An action is a decision to switch or maintain the placement plan. The cost model captures the effect of the action on the end-to-end latency that the application perceives and/or the energy consumed by the used resources. Each of the placement plans in the acquired policy is deployed on to the host network such that they can be *activated* depending on the current application and system state by following the policy.

Therefore, this work makes the following contributions:

- The problem of placing DSP application tasks across edge and cloud computing resources is modelled as an infinite horizon discrete time MDP with discounted rewards.
- The work introduces a novel methodology to reduce the state space of the MDP by using a two phase approach where only the locally optimal placement plans are considered with respect to identified different environment states which drastically reduces the size of the state space.
- We introduce a configurable reward function that captures the current state of the task placement and the next possible placements to achieve an optimal task placement.
- We show the accuracy of TSTP by comparing it against a greedy task placement strategy as well as multiple static task placement strategies for one synthetic application and two real world applications.

The rest of this work is organised as follows. We introduce the task placement problem in Section 5.2 followed by the TSTP formulation in Section 5.3. The experiments and the results are discussed in Section 5.4 and the related work in Section 5.5 where the Section 5.6 concludes the study.

5.2 Background

As described in Section 3.2.2, a DSP application is often structured as a dataflow whose tasks comprising *sources*, *operators* and *sinks*, and the streams between tasks that allow a tuple/message to flow from one task to another for processing. Figure 5.1 depicts a simple application dataflow comprising a single source (S_0), two operators (Op_1 and Op_2) and a sink (S_i).

FIGURE 5.1: Simple data stream processing application with one source, two operators and one sink.

A DSP application is generally deployed on a heterogeneous connected network of resources. We consider an infrastructure that contains both cloud and edge resources, each with their own characteristics. Prior to deployment, each application task is mapped to one or more of these resources of the underlying host network. The process of mapping a task to a host node is known as task placement. The task placement affects the overall Quality of Service (QoS) delivered by the application which is often measured by metrics such as the mean end-to-end latency of event processing or the mean application throughput. Our work in Chapter 4 has shown that it is possible to derive one or more optimal task placement plans for an application, based on the host network upon which it is deployed. The optimality criteria are achieved based on an objective function developed to measure one or more metrics, such as the latency or energy consumption.

However, the optimal task placement is tightly coupled with a particular configuration of the host network and the application. Once the tasks are placed, the conditions that affect the application execution, and in turn the measured metrics, may vary depending on the characteristics of the resources and those of the application. CPU, memory, storage and power consumption vary depending on the underlying resources. For example, an edge device may reduce the CPU clock speed if the device is operating in a low power mode due to a drained battery. The event generation rate (*i.e.*, rate at which sources generate events) and the event sizes vary depending on the application. For instance, an application based on monitoring city traffic may observe increased activity during certain times of the day or week. We denote as environment states the changes that occur on the application, and computing and network infrastructure.

Our work introduced in Chapter 4 shows a constraint optimisation model that finds an initial optimal task placement plan for a given environment, assuming that it remains static. Nevertheless, in many real world scenarios the environment conditions can vary during the application runtime, so that the initial placement plan is no longer optimal. Depending on the current environment state, the optimal task placement needs to be re-evaluated and redeployed. However, changing the placement of DSP tasks that continuously process a stream of incoming events is not trivial as it incurs considerable overhead and impacts the system performance. For example, a common approach for changing an initial task placement consists of moving tasks from one resource to another. For such, the application is first stopped, then the task states (stored buffers and files) are transferred to a new resource, and finally the application is restarted. Any

pending events are then replayed before resuming the processing of incoming events. Consequently, frequent changes in task placement of this nature are not ideal where near real-time processing is required.

To overcome the issue of relying on the initial static task placement over the entire application life-cycle and to reduce the overhead of frequent task placement changes, we propose a two-phase task placement strategy modelled as an MDP.

5.3 Problem Formulation

A Markov Decision Process consists of states, actions, a probability function that defines how the system transitions between states, and a reward function associated with each transition. We consider each of them here and formulate the problem as an infinite horizon MDP. The notation used throughout this study is summarised in Table 5.1.

We explain the model using the simple linear DSP application topology depicted in Figure 5.1. For the sake of illustration, we consider that the DSP application is deployed on an infrastructure consisting of two resources; one cloud node and one edge node. For environment states, the event generation rate takes only one of the following discrete values: low, medium (med) or high. The probability of transitioning between environment states is $\{P_{e,e'}|e, e' \in \{low, med, high\}\}$ and is assumed to be known.

5.3.1 States

Let $\mathcal{E}$ be the set of possible states of the environment conditions, such as the current event generation rate, event size, and the load on the system with $|\mathcal{E}| = k$. Let $\mathcal{M}$ be the set of optimal placement plans which we have acquired from our work described on Chapter 4 for each environment condition $e \in \mathcal{E}$ with $|\mathcal{M}| = l$. Here, for a given environment state, one or more optimal placement plans may be available. If more than one plan is available for a given environment state, one optimal placement plan needs to be selected based on an application policy such as choosing the policy that results in highest utilisation of resources. In each placement plan $m \in \mathcal{M}$ each task of the DSP application topology is mapped to a resource in the host network. The set of states $\mathcal{S}$ here is the finite set of tuples such that $\mathcal{S} = \mathcal{E} \times \mathcal{M}$ and $\mathcal{S} = \{s \mid s = \langle e, m \rangle, \forall e \in \mathcal{E}, \forall m \in \mathcal{M}\}$, where $|\mathcal{S}| = k \times l$.

Without loss of generalisation, we define $\mathcal{S}^* \subset \mathcal{S}$ as the set containing the optimal plans with respect to the corresponding environment state. Let F be the real valued (minimising) objective function listed in Chapter 4 to acquire the optimal placement

TABLE 5.1: Notations used in the formulation of TSTP.

Symbol	Description
$\mathcal{E}$	Set of possible states of the environment conditions
P	Probability of transitioning between environment states
k	Number of environment states
$\mathcal{M}$	Set of optimal placement plans
l	Number of optimal placement plans
F	Real valued objective function used for obtaining each optimal placement plan
$\mathcal{S}$	All combinations of environment states and optimal placement plans
$\mathcal{S}^*$	Set containing the corresponding optimal placement for each environment state
$\mathcal{S}^\#$	Set containing the least desired placement plan for the corresponding environment state
$\mathcal{A}$	Set of actions
T	Transition probability function
R	Real valued reward function
$\Phi_X(s)$	Real valued function that measures the metric X for deployment of state $s \in \mathcal{S}$ on the resources
$L(s, a, s')$	Per-event total latency reward gained by transitioning from state s to s' due to action a
$E(s, a, s')$	Per-event energy consumption reward of the edge resources gained by transitioning from state s to s' due to action a
C_{X_1} and C_{X_2}	Non-zero real constants that affect the shape of the reward function
$cost_X(s, a, s')$	Real valued cost of transitioning from state s to s' due to action a measured by the metric X
α	Unit interval used for calculating the reward as a weighted sum
π	A deterministic stationary policy generated by MDP
π^*	Optimal policy generated by MDP
γ^t	Discount factor for the long term reward

plan(s) given the environment state, i.e., $F : \mathcal{E} \times \mathcal{M} \to \mathbb{R}$. For $\langle e, m\rangle \in \mathcal{S}^*$, $F(e, m) \leq F(e, m'), \forall m' \in \mathcal{M}$. Similarly, we also define the set $\mathcal{S}^\# \subset \mathcal{S}$ as the set containing the least desired plan with respect to the environment state of the state, i.e., for $\langle e, m\rangle \in \mathcal{S}^\#$, $F(e, m) \geq F(e, m'), \forall m' \in \mathcal{M}$.

Using this model, the state space of the example application in Figure 5.1 can be derived in the following manner:

- $\mathcal{E} = \{low, med, high\}$
- Let $\mathcal{M} = \{m_0, m_1, m_2\}$ where, $\mathcal{S}^* = \{\langle low, m_0\rangle, \langle med, m_1\rangle, \langle high, m_2\rangle\}$

- $m_0 = \langle 0, 0, 0, 1 \rangle, m_1 = \langle 0, 0, 1, 1 \rangle, m_2 = \langle 0, 1, 1, 1 \rangle$, where 0 denotes an edge placement and 1 denotes a cloud placement.
- Therefore, $\mathcal{S} = \{\langle low, m_0 \rangle, \langle low, m_1 \rangle, \langle low, m_2 \rangle, \langle med,\ m_0 \rangle, \langle med, m_1 \rangle, \langle med, m_2 \rangle, \langle high, m_0 \rangle, \langle high, m_1 \rangle, \langle high, m_2 \rangle\}$

5.3.2 Actions

The set of actions $\mathcal{A}$ is defined in terms of the next plan that (stochastically) results from taking an action. It is a finite set such that $\mathcal{A} = \{a_0, a_1, a_2, \ldots, a_l\}$ and $|\mathcal{A}| = |\mathcal{M}|$. In each state, there are exactly $|\mathcal{M}|$ actions since each action selects one of the available plans, *i.e.* here, the only options are to transition to one of the $|\mathcal{M}| - 1$ other placement plans or to stay on the current placement plan.

As a result, for the example application the available actions are:

- $\mathcal{A} = \{a_0, a_1, a_2\}$

5.3.3 Discrete Time Sampling

In order to define when an action is executed, we define a discrete global clock, $t = 0, 1, 2, \ldots$, with a period δ. We assume a Markovian model for transitioning between the states in our application since the decision to transition to another state depends only on the current state:

$$Pr(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \ldots) = Pr(s_{t+1}|s_t, a_t)$$

5.3.4 Transition Probabilities

By applying action $a \in \mathcal{A}$ in a state $s \in \mathcal{S}$, a transition is made from s to a new state $s' \in \mathcal{S}$, based on a probability distribution over the set of possible transitions. The transition function T is defined as $T(s, a, s')$ where $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$. In addition, $\sum_{s' \in S} T(s, a, s') = 1$.

In our MDP, part of the state is determined by the action, as shown in Figure 5.2, and so for $s = \langle e, m \rangle, a = a_{m'}$ and thereby $s' = \langle e', m' \rangle$ we have only probability $p_{e,e'}$ to consider such that $T(s, a, s') = p_{e,e'}$.

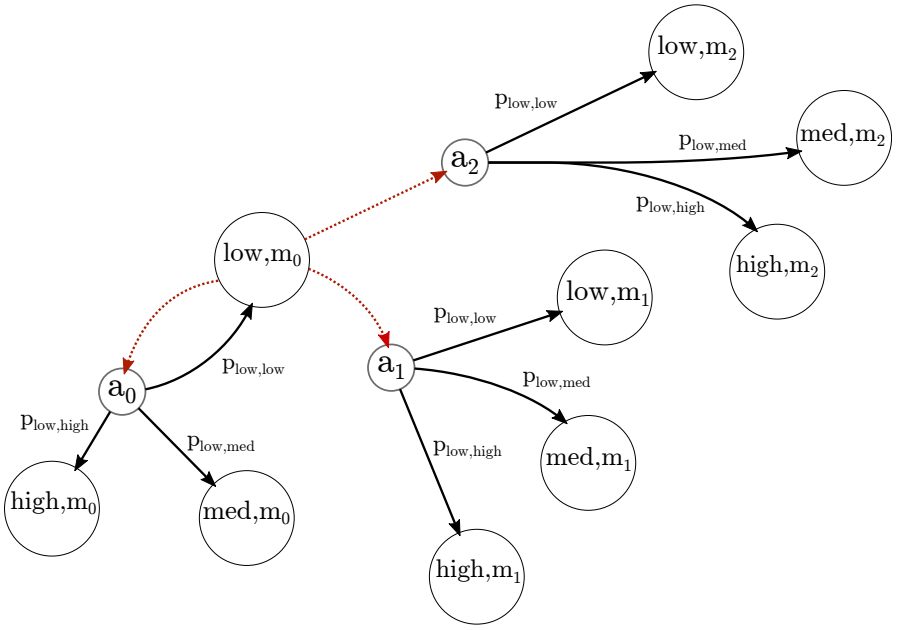


FIGURE 5.2: Partial state transition diagram for all the actions available from state $\langle low, m_0 \rangle$.

5.3.5 Reward Function

The reward function defines the reward gained for performing a certain action in a given state which results in transitioning to another (or the same) state. We define the state reward function R as a real valued function such that $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$.

Let, $R(s, a, s')$ be the reward at s' after transitioning from s due to a, and $s = \langle e, m \rangle$, $s' = \langle e', m' \rangle$, $s^* = \langle e, m^* \rangle$, $s'^* = \langle e', m'^* \rangle$, $s^\# = \langle e, m^\# \rangle$, $s'^\# = \langle e', m'^\# \rangle$. Function Φ_X is a real valued function such that $\Phi_X : \mathcal{S} \rightarrow \mathbb{R}$ and is used to calculate the metric X for the deployment of placement plan at a given state.

We consider latency ($L \in X$) and energy consumption ($E \in X$) as metrics so that $L(s, a, s')$ denotes the latency reward and $E(s, a, s')$ denotes the energy reward derived from $X(s, a, s')$ in Equation (5.6) formulated below.

The latency or energy gain achieved in the next state against the optimal placement plan with respect to the maximum possible gain in the next state can be calculated as:

$$x_0 = \frac{\Phi_X(s'^*) - \Phi_X(s')}{\Phi_X(s'^\#) - \Phi_X(s'^*)} \tag{5.1}$$

Similarly, the absolute latency or energy gain when compared with the optimal placement plan in the current state with respect to the maximum possible gain in the current state is calculated as:

$$x_1 = \frac{|\Phi_X(s^*) - \Phi_X(s)|}{\Phi_X(s^\#) - \Phi_X(s^*)} \tag{5.2}$$

The normalised latency or energy of the next state is given by:

$$x_2 = \frac{\Phi_X(s')}{\sum_{\forall m_i} \Phi_X(\langle e', m_i \rangle)} \tag{5.3}$$

We use this function to acquire a sense of the latency or energy at the next state relative to other placement plans available under the same environment state.

The following dampener function is used to reduce the reward of the next placement plans that are further away from the optimal placement plans:

$$x_3 = \frac{C_{X_1}}{1 + e^{C_{X_2} x_0}} \tag{5.4}$$

Here, C_{X_1} and C_{X_2} are non-zero real constants that affect the shape of the reward function.

Equations (5.1) to (5.4) are combined in the following manner to produce a function such that as the plan migration gets closer to a migration from an optimal placement plan to another optimal placement plan, the value of x_4 approaches zero.

$$x_4 = \frac{x_0 \times x_2 \times x_3}{1 + x_1} \tag{5.5}$$

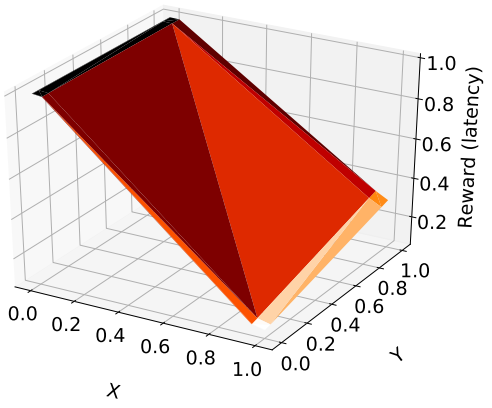
x_4 is then used in Equation (5.6) to acquire the final reward value where a migration from optimal placement plan to another optimal placement plan has a value of 1 and other sub or non optimal migrations have a value less than 1 with respect to their optimality[1].

$$X(s, a, s') = \frac{1}{1 + (x_4)^2} - cost_X(s, a, s') \tag{5.6}$$

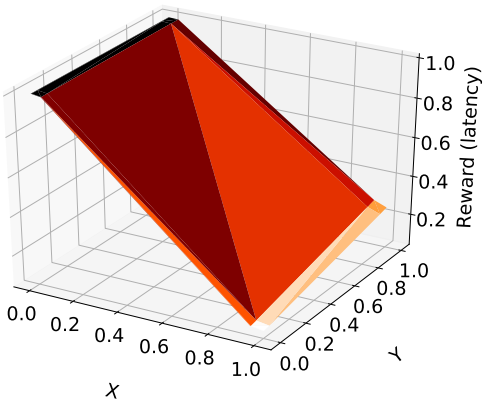
where $cost_X(s, a, s')$ denotes a real valued function that calculates the cost of transitioning with respect to the metric X caused by taking the action a which includes the overhead of transferring operator states from s to s'.

Figure 5.3 illustrates the distribution of the reward function formed in Equation (5.6) with latency measurements considered for the Simple application described earlier as shown in Figure 5.1. The plots show that an aggressive reward function where staying on an optimal state as well as migrating to an optimal state is highly rewarded, can be devised by having higher C_{X_1} configurations. Higher C_{X_2} configurations reduce the reward given to a migration from a current optimal placement plan to another non-optimal placement and the reward reduction increases with the value of C_{X_2} if the next placement plan is not optimal.

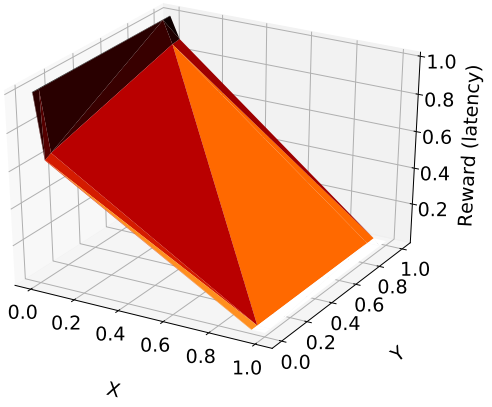
[1] or $1 - cost_X(s, a, s')$ if a cost is involved.



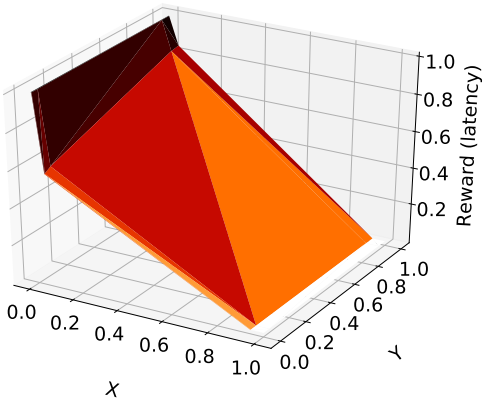
(a) $C_{L_1} = 10$ and $C_{L_2} = 2$



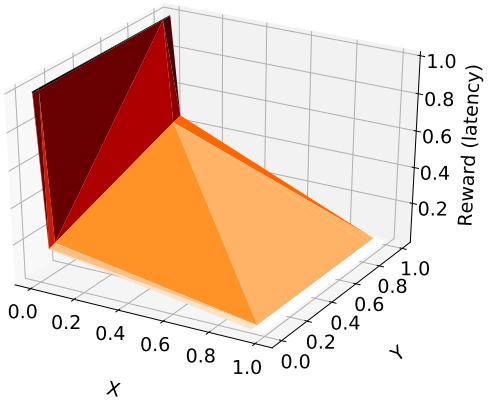
(b) $C_{L_1} = 10$ and $C_{L_2} = 10$



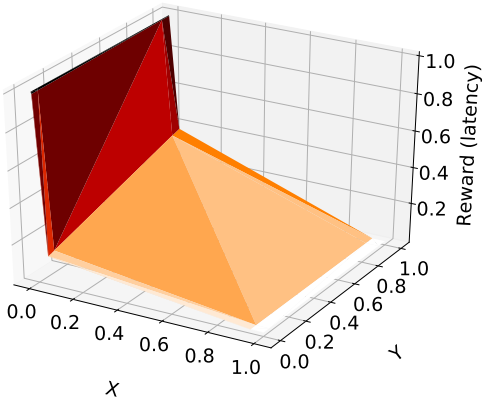
(c) $C_{L_1} = 100$ and $C_{L_2} = 2$



(d) $C_{L_1} = 100$ and $C_{L_2} = 10$



(e) $C_{L_1} = 300$ and $C_{L_2} = 2$



(f) $C_{L_1} = 300$ and $C_{L_2} = 10$

FIGURE 5.3: Distribution of the latency reward for the Simple application illustrated in Figure 5.1. X axis represents the normalised latency difference between the next state and the optimal state. Y axis represents normalised latency difference between the current state and the optimal state. Darker colors represent higher rewards.

The total reward function R is composed as a weighted sum of the above latency reward and energy reward. Given a constant α where $0 \leq \alpha \leq 1$:

$$R(s, a, s') = \alpha L(s, a, s') + (1 - \alpha)E(s, a, s') \tag{5.7}$$

5.3.6 Policies

A policy for a MDP is a function that takes a state as an input and outputs an action for the state. We consider a deterministic policy π as a function $\pi : \mathcal{S} \rightarrow \mathcal{A}$. Initially, given a starting state s_0, the policy outputs an action $a_0 = \pi(s_0)$ which is then performed by the system. Once the action is performed, given a transition function T and a reward function R, a transition is made to state s_1 with the probability $T(s_0, a_0, s_1)$ which results in achieving the reward r_0. This process is continued indefinitely such that $s_0, \pi(s_0), r_0, s_1, \pi(s_1), r_1, s_2, \pi(s_2), r_2, \ldots$ are produced and applied.

We consider a stationary policy, *i.e.,* a policy that outputs the same action regardless of the time, since for an infinite horizon discounted cost MDP, the optimal solution is acquired through a stationary policy [260].

5.3.7 Optimality Criteria

Since we model our MDP as an infinite horizon problem, the long term reward needs to be taken in to account in deciding the optimality criteria. Given the reward function R at decision epoch t, and a discount factor γ^t where $0 < \gamma < 1$ we try to optimise the expected return:

$$\mathbb{E}\left(\sum_{t=1}^{\infty} \gamma^t R_t\right) \tag{5.8}$$

5.3.8 Value Function

Here, we present the value function which links the optimality criteria to the policies. Let $V^{\pi}(s)$ be the value of a state s under policy π which is the expected return when starting at state s and following the policy π thereafter. Using Equation (5.8), $V^{\pi}(s)$ can be defined for an infinite horizon discounted reward MDP as:

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left\{\sum_{i=0}^{\infty} \gamma^i R_{t+i} | s_t = s\right\} \tag{5.9}$$

Since these value functions satisfy recursion, these equations can be rewritten as a Bellman equation [261].

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}_{\pi}\left\{R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \ldots | s_t = s\right\} \\ &= \mathbb{E}_{\pi}\left\{R_t + \gamma V^{\pi}(s_{t+1}|s_t = s)\right\} \\ &= \sum_{s'} T(s, \pi(s), s')\left(R(s, a, s') + \gamma V^{\pi}(s')\right) \end{aligned} \tag{5.10}$$

The goal for any given MDP is to find the optimal policy that maximises the reward which translates to maximising the value function Equation (5.9) $\forall s \in \mathcal{S}$. An optimal policy is denoted as π^* and $V^{\pi^*}(s) \geq V^{\pi}(s)$. For the optimal policy we can acquire the Bellman optimality equation as:

$$V^{\pi^*}(s) = \max_{a \in A} \sum_{s' \in S} T(s, a, s')\left(R(s, a, s') + \gamma V^{\pi^*}(s')\right) \tag{5.11}$$

and to acquire the optimal action given the optimal value function $V^{\pi^*}(s)$ the following equation can be used:

$$\pi^*(s) = \arg\max_{a \in A} \sum_{s' \in S} T(s, a, s')\left(R(s, a, s') + \gamma V^{\pi^*}(s')\right) \tag{5.12}$$

Since the problem is formulated as a model-based MDP, we assume that the transition functions T and reward functions R are known. We solve the model based MDP using the value iteration algorithm [261][2] in multiple iterations.

5.4 Performance Evaluation

The experiments are performed using our ECSNeT++ simulator[3] introduced in Chapter 3, specifically designed to model and simulate data stream processing applications on heterogeneous edge-cloud infrastructure which has also been verified against real device deployments. Resources are modelled as two Raspberry Pi 3 Model B devices (as edge devices Pi1 and Pi2) connected via separate access points to a single cloud node (C1) as illustrated in Figure 5.4.

[2]We use the python MDP toolbox library to acquire our solutions. (Available at https://pymdptoolbox.readthedocs.io/en/latest/api/mdp.html.)

[3]Available at: https://sedgecloud.github.io/ECSNeTpp/

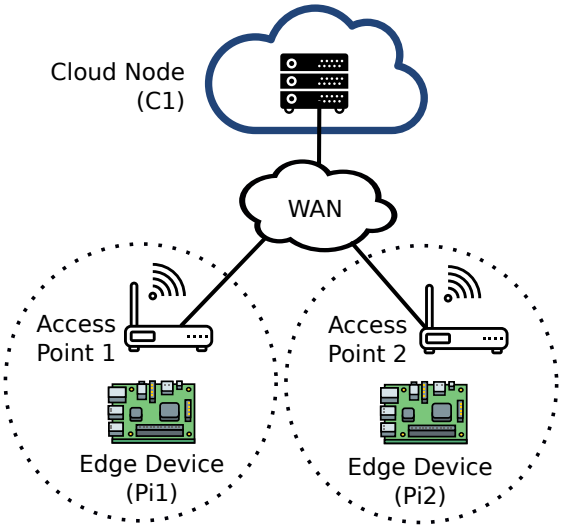


FIGURE 5.4: Simulated edge-cloud infrastructure used for executing the applications.

5.4.1 Policies

In order to compare the optimality of TSTP, we compare it with two other task placement policies that can be applied to the task placement problem on edge-cloud infrastructure, namely Greedy Policy and Resource-aware Placement Policy [81], and the set of locally optimal static task placement plans acquired in the first phase.

5.4.1.1 TSTP Policy

The TSTP policy uses the proposed two-phase MDP based approach as outlined below.

1. First Phase - Identify the locally optimal placement plans
   (a) Analyse the application and the anticipated environment conditions (such as the event generation rate changes) to acquire the environment states and the probability of transitioning between the states. ($\mathcal{E}$ and T)
   (b) For each identified environment state, acquire the optimal task placement plan. If more than one optimal task placement plan is available, select one placement plan based on the application/environment/end-user policy. ($\mathcal{M}$ and $\mathcal{S}'$)
   (c) Deploy each task placement individually on the infrastructure and measure the latency (and other metrics used for calculating the rewards) for each environment state identified in the first step.
2. Second Phase - Acquire the global MDP policy for switching between the optimal placement plans based on varying environment conditions

(a) Use the optimal placement plans, environment states, and metrics acquired in the first phase, define $\mathcal{S}$, $\mathcal{A}$, T and R to formulate the MDP.

(b) Solve the MDP to acquire the stationary TSTP policy.

(c) Deploy the identified placement plans in the policy onto the resource infrastructure.

(d) Use the policy to switch between the deployed placement plans depending on the monitored current environment condition.

5.4.1.2 Greedy Policy

The greedy policy is implemented as a memoryless policy to select the locally optimal policy out of those acquired in the first phase of the TSTP methodology for the current environment state while ignoring any future reward.

When executing either the TSTP or the greedy policy, the locally optimal placement plans have already been deployed on to devices and the deployed plans are *activated* according to the policy to process the incoming events. We assume that this cost of initially deploying all the locally optimal placement plans onto devices, is negligible due to the long running nature of DSP applications.

5.4.1.3 Resource-aware Placement Policy (RA)

Taneja and Davy [81] present a resource aware static task placement policy which attempts to satisfy the requirements of the given DSP application using the CPU, memory and bandwidth capacity of the edge and cloud infrastructure. For the RA policy, the placement plan remains static.

5.4.2 Applications

To show the versatility of the MDP model for task placement on varying environment conditions, we consider a simple synthetic DSP application topology as illustrated in Figure 5.1 and two real world applications from the RIoTBench suite [1]; STATS application and ETL+STATS application, as depicted in Figure 5.5 and Figure 5.6 respectively. For all three of these applications, the following assumptions are made during task placement:

- Sources are placed on one of the edge nodes.

- Sinks are placed on the cloud node.
- Other operators can be placed on either a cloud or edge node.

In addition, due to the requirements of the STATS application and the ETL+STATS application, the SenML Parser task is also placed on the same edge node in which the respective source is placed.

5.4.2.1 Simple DSP application

The simple application has a linear topology with one source, one sink and 2 operator tasks – see Figure 5.1. The operator properties are shown in Table 5.5. As modelled in Section 5.3, our MDP has the following properties:

- $\mathcal{E} = \{low, med, high\}$
- $\mathcal{M} = \{Plan\ 0, Plan\ 1, Plan\ 2\}$
- $\mathcal{S} = \{s_i | s_i = \langle e, m \rangle\ \forall e \in \mathcal{E}, \forall m \in \mathcal{M}\}$
- $\mathcal{A} = \{a_0, a_1, a_2\}$
- $\gamma = 0.96,\ \alpha = 1$
- $C_{L_1} = 100, C_{L_2} = 2$[4]
- $cost_L(s, a, s') = 0$

The assumptions made earlier regarding task placement which further reduce the search space, resulted in three possible placement plans for each environment condition in the first phase. Our constraint optimisation framework suggested to utilise only one edge node along with the cloud node for this application. Then we first deploy each acquired placement plan on the experimental set up individually and execute them to process 10000 events with the event generation rates identified as the three environment conditions to acquire the latency measurements for each environment state for each plan as shown in Table 5.2. The latency values are obtained by deploying each plan on the experimental set up with event generation rate following a normal distribution with the mean of either 10, 100 or 200 events per second. The optimal latency value for each environment condition (*i.e.* event generation rate) is highlighted in bold in the table. These latency values are then used in the MDP reward function to calculate the rewards for each action.

[4]Values were obtained after running multiple simulations.

TABLE 5.2: Acquired placement plans for the simple application and their measured latency values.

Plan	Placement	Mean Latency (ms)		
		low (10evps)	med (100evps)	high (200evps)
0	$\langle Pi1, Pi1, Pi1, C1\rangle$	238.62	**187.58**	174.96
1	$\langle Pi1, Pi1, C1, C1\rangle$	240.07	189.67	**174.82**
2	$\langle Pi1, C1, C1, C1\rangle$	**214.38**	234.54	233.01

5.4.2.2 STATS Application

Next we consider the STATS application from the RIoTBench suite [1] as illustrated in Figure 5.5 which extracts statistical information from a stream of events. The operator properties of the STATS application are shown in Table 3.4 and the following MDP model is formulated to acquire the optimal policy for the identified environment states:

- $\mathcal{E} = \{1evps, 10evps, 20evps, 50evps\}$
- $\mathcal{M} = \{Plan\ 0, Plan\ 1, Plan\ 3, Plan\ 6\}$
- $\mathcal{S} = \{s_i | s_i = \langle e, m\rangle\ \forall e \in \mathcal{E}, \forall m \in \mathcal{M}\}$
- $\mathcal{A} = \{a_0, a_1, a_2, a_3\}$
- $\gamma = 0.96,\ \alpha = 0.8$
- $C_{L_1} = 400, C_{L_2} = 2, C_{E_1} = 400, C_{E_2} = 2^4$
- $cost_L(s, a, s') = 0.005$

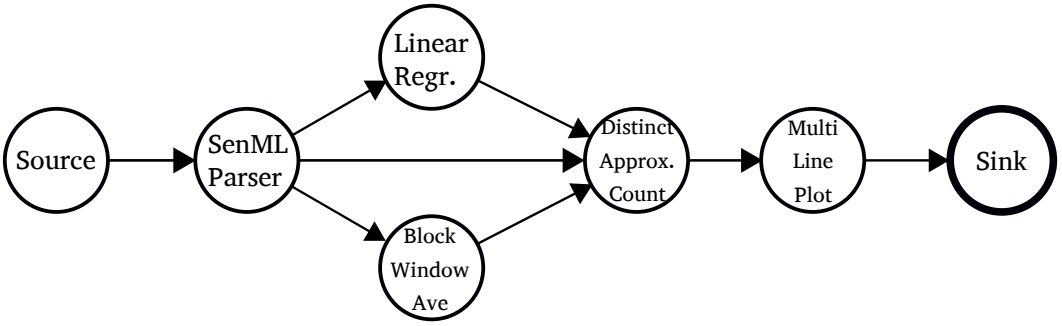


FIGURE 5.5: STATS application topology from the RIoTBench suite.

Similar to the Simple application, we acquire the latency values as shown in Table 5.3 which are then used in the MDP model to acquire the optimal stationary policy. The latencies are obtained by deploying each plan on the experimental set up with event generation rate following a normal distribution with the mean of either 1, 10, 20 or 50 events per second. The optimal latency value for each environment condition is highlighted in bold in the table.

TABLE 5.3: Acquired placement plans for STATS application and their measured latency values.

Plan	Placement	Mean Latency (ms)			
		1evps	10evps	20evps	50evps
0	$\langle Pi1, Pi1, Pi1, Pi1, Pi1, Pi1, C1\rangle$	848.23	949.75	969.01	**974.06**
1	$\langle Pi1, Pi1, Pi1, Pi1, Pi1, C1, C1\rangle$	**393.52**	382.84	498.25	174658.88
3	$\langle Pi1, Pi1, Pi1, C1, C1, C1, C1\rangle$	403.15	380.38	**449.69**	27856.23
6	$\langle Pi1, Pi1, Pi2, C1, C1, C1, C1\rangle$	398.04	**375.86**	456.21	16949.91

5.4.2.3 ETL+STATS Application

In order to analyse how a large scale realistic application with a large number of tasks affects the resolution time, we consider the ETL+STATS application from the RIoT-Bench suite [1]. As illustrated in Figure 5.6, the application consists of 1 source, 12 operators and 2 sinks where part of the application is the STATS application discussed in Section 5.4.2.2 with the operator configurations as depicted in Table 3.4. The MDP formulation of this application is as follows.

- $\mathcal{E} = \{5evps, 10evps, 20evps, 50evps, 100evps\}$
- $\mathcal{M} = \{Plan\ 4-6, Plan\ 5-5, Plan\ 6-3, Plan\ 6-7\}$
- $\mathcal{S} = \{s_i | s_i = \langle e, m\rangle\ \forall e \in \mathcal{E}, \forall m \in \mathcal{M}\}$
- $\mathcal{A} = \{a_0, a_1, a_2, a_3\}$
- $\gamma = 0.98,\ \alpha = 1$
- $C_{L_1} = 126, C_{L_2} = 10^4$
- $cost_L(s, a, s') = 0.01$

For each of the identified environment conditions, out of the 42 possible placement plans, 4 placement plans were identified in the first phase. The latency values of the identified placement plans are shown in Table 5.4.

We have set the per-event latency cost on switching between plans to 0.005 s and 0.01 s for STATS and ETL+STATS applications respectively. While we conduct our experiments using ECSNeT++ with the transfer cost configured, studies such as Merlin [208] introduce reconfiguration of task placement with minimum transfer cost which we can utilise for TSTP as well.

TABLE 5.4: Acquired placement plans for ETL+STATS application and their measured latency values.

Plan	Placement		Mean Latency (ms)				
	Edge	Cloud	5evps	10evps	20evps	50evps	100evps
Plan $4-6$	so, pa1, rf, bf, ip, jo, an, pa2, lr, bwa	csv, si1, dac, mlp, si2	307.22	357.47	**462.11**	61397.24	91015.83
Plan $5-5$	so, pa1, rf, bf, ip, jo, pa2, lr, bwa, dac	an, csv, si1, mlp, si2	**270.7**	**302.79**	488.39	97337.82	155653.5
Plan $6-3$	so, pa1, rf, bf, pa2, lr, bwa, dac, mlp	ip, jo, an, csv, si1, si2	644.79	593.54	571.45	**554.65**	10655.39
Plan $6-7$	so, pa1, rf, bf, ip, jo, an, csv, pa2, lr, bwa, dac, mlp	si1, si2	665.01	638.11	591.66	560.99	**550.04**

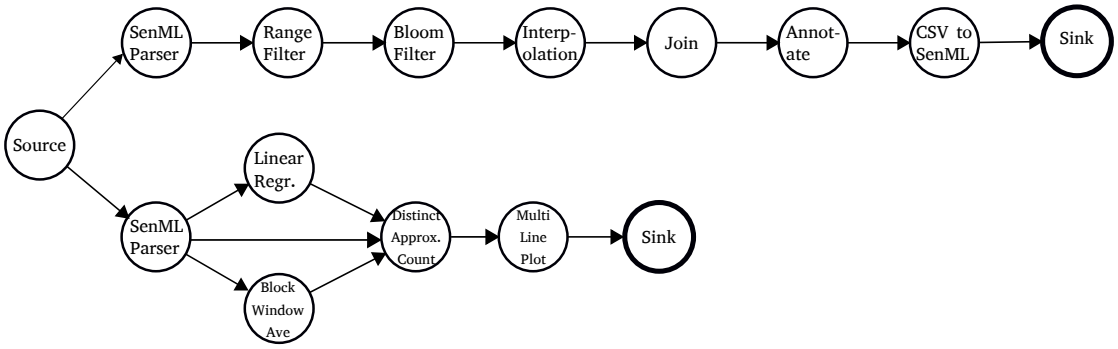


FIGURE 5.6: ETL+STATS application topology from the RIoTBench suite.

TABLE 5.5: Properties of the operators of the applications used for evaluation of TSTP.

Application(s)	Operator	Characteristic	
		Selectivity	Productivity
Simple	Op_1	10	1
	Op_2	1	0.2

5.4.3 Performance Evaluation Results

In order to analyse the performance of TSTP against other policies we first take a detailed look at the performance of the STATS application with different task placement policies on varying environment conditions. Figure 5.7 illustrates time-lines of three application scenarios of the STATS application and Figure 5.8 depicts the mean per-event latency composed by the processing and network latencies of the STATS application in these scenarios.

Figure 5.7(a) shows how the latency measurements change when the event generation rate is gradually increased within the identified four discrete event generation rates. Figure 5.7(b) shows the behaviour of STATS application amidst bursts of high event generation rates. Figure 5.7(c) illustrates a bi-modal distribution of the event generation rate similar to human behaviour where during mornings and evenings a high number of events are generated by devices such as fitness trackers. Overall in all three scenarios, for mean per-event latency, TSTP outperforms the dynamic greedy policy as well as the other static policies. Due to the application complexity and the placement plans, TSTP was able to produce a policy that maximises the future rewards (*i.e.* value) by considering the transition probabilities when compared to the memoryless greedy approach. It is also important to note that when the event generation rates increase, the static plans 1, 3 and 6 cause the system to become unstable. As a result the per-event latency grows significantly due to the network being congested with the high number of events emitted by the tasks as evident in the high network latencies illustrated in Figure 5.8. However, the TSTP policy does not suffer from this effect since it switches to the stable Plan-0 during high event rates and remains in the stable Plan-6 in the low

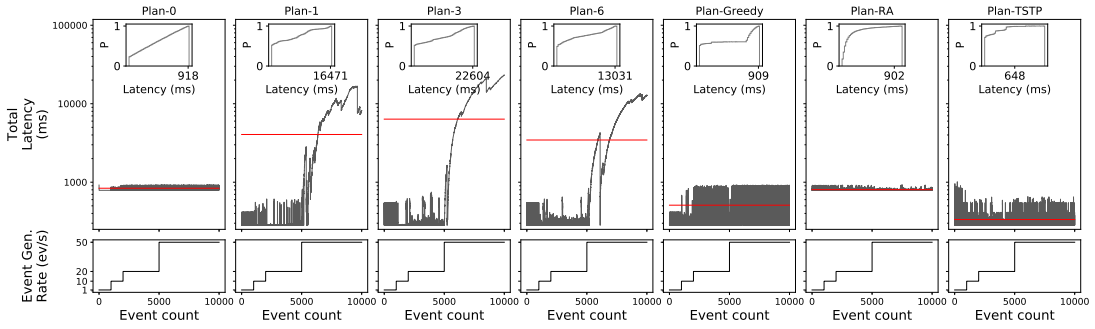
event rates. As evident in Table 5.3, Plan-0 produces low latency results in high event rates whereas Plan-6 consistently produces low latency results in low event rates.

Each sub-plot within the per-event latency time-line plots of Figure 5.7, illustrates the cumulative density function of the per-event latency and the latency of the 99^{th} percentile. Here, for bursty and bi-modal event generation scenarios, we can observe that the resource aware task placement policy has resulted in a lower or equal latency to TSTP in the 99^{th} percentile. However, by observing the distribution of the latency line plots in Figure 5.7(b) and Figure 5.7(c) as well as by observing the mean per-event latencies presented in Figure 5.8(b) and Figure 5.8(c), it is evident that the RA policy produces higher per-event latencies on average as opposed to the lower per-event latencies produced by TSTP.

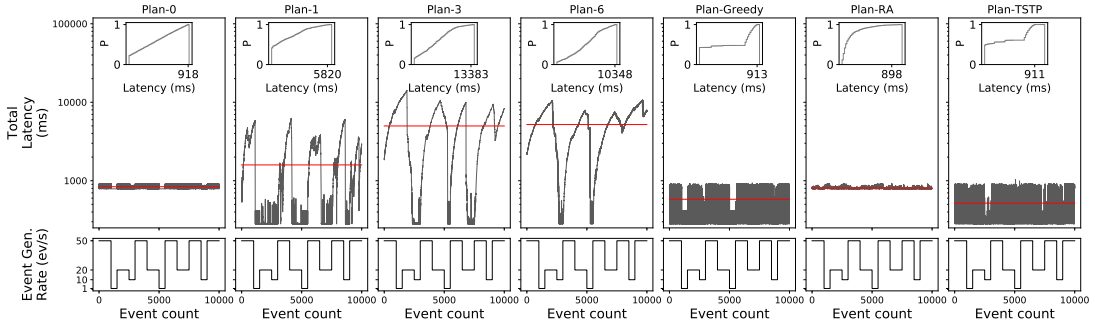
The results for the simple application considering four distinct event generation rates are presented in Figure 5.9. Figure 5.9(a) shows how the Simple application behaves with regard to different placement policies amidst bursty event generation. When the event generation rate changes frequently, TSTP is able to continuously outperform the greedy policy as well as the static placement policies. The TSTP achieves an average latency reduction of 3.49 ms per event for the bi-modal event generation as shown in Figure 5.9(a) and 1.15 ms for the bursty event generation scenario as shown in Figure 5.9(b) when compared to the greedy policy. TSTP also suggests to oscillate between the Plan-0 and Plan-2 when the mean event generation rate is above 200evps whereas greedy policy remained stationary at Plan-1.

Similarly, Figure 5.10 shows the latency measurements for the ETL+STATS application. Here, in both scenarios the TSTP outperforms the other placement policies where the mean latency difference to the greedy policy is 30.38 ms per event for the bursty event generation and 40.28 ms for the bi-modal generation when compared with the greedy placement policy. Here, the TSTP policy oscillates between Plan 4-6 and Plan 6-7. As evident here as well as in the STATS application results, when the application becomes complex to navigate, we can observe that TSTP is able to incorporate the inherent characteristics of the application such as considering placement plans that produce low latency results for given environment conditions and the probability of observing these event rates in the future, and utilise that to produce an optimal policy that outperforms other policies in the long run.

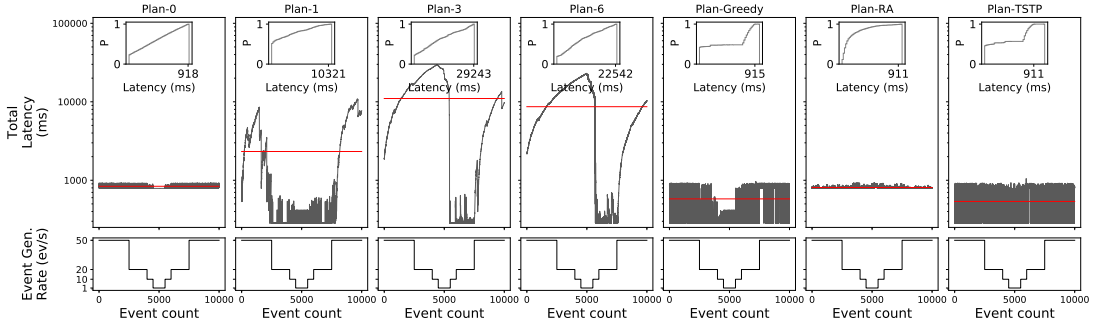
In both Simple application and the ETL+STATS application in all of these scenarios, TSTP outperforms the Resource-Aware task placement since it does not take into account the latency minimisation requirements of the DSP applicaiton.



(a) Latency measured for gradually increasing event generation rates



(b) Latency measured for event generation with bursts of high event rates

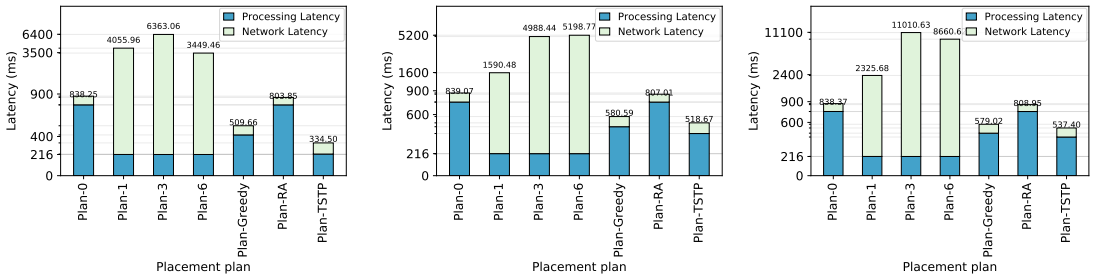


(c) Latency measured for event generation in a bi-modal distribution

FIGURE 5.7: Time-lines of latency measurements for processing 10000 events of the STATS application in 5 static placement plans (Plan 0, 1, 3, 6, and RA), greedy placement plan and TSTP placement plan when the mean of the event generation rate is changing between 1 evps, 10 evps, 20 evps and 50 evps as plotted in the bottom row of each scenario. The red line in each plot illustrates the mean per-event latency for comparison. The sub-plot within each latency plot illustrates the cumulative density function of the per-event latency and the value shown on the x-axis is the per-event latency of the 99^{th} percentile.

5.4.4 Mitigating State Explosion in TSTP

The main objective of introducing our TSTP approach was to reduce the state explosion inherent to MDP problems. To achieve that goal, we reduced the states in the first phase by pruning the non-optimal placement plans with respect to identified environment states.

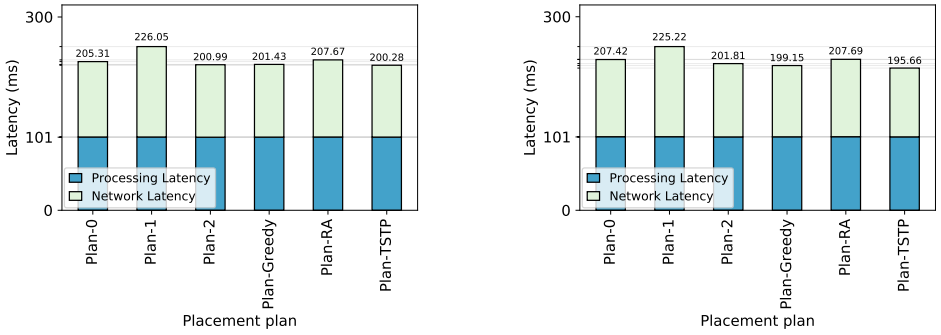


(a) Latency measured for gradually increasing event generation rates

(b) Latency measured for event generation with bursts of high event rate

(c) Latency measured for event generation in a bi-modal distribution

FIGURE 5.8: Summarised mean per-event latency (composed as the sum of processing and network latencies) for processing 10000 events of the STATS application in 5 static placement plans (Plan 0, 1, 3, 6, and RA), greedy placement plan and TSTP placement plan when the mean of the event generation rate is changing between 1evps, 10evps, 20evps and 50evps. Each bar is annotated with the mean per-event latency value for each placement policy. Y-axis is in the symmetric log scale.



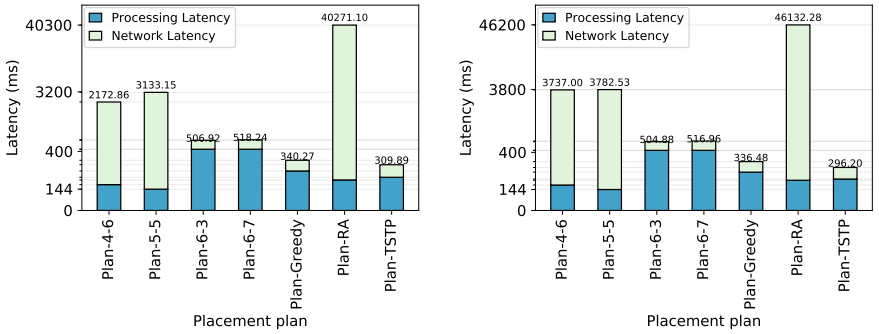
(a) Latency measured for event generation with bursts of high event rates

(b) Latency measured for event generation in a bi-modal distribution

FIGURE 5.9: Summarised mean per-event latency (composed as the sum of processing and network latencies) for processing 50000 events of the Simple application in 4 static placement plans (Plan 0, 1, 2, and RA), greedy placement plan and TSTP placement plan when the mean of the event generation rate is changing between 10evps, 100evps and 200evps. Each bar is annotated with the mean per-event latency value for each placement policy. Y-axis is in the symmetric log scale.

In the Value Iteration algorithm, each iteration can be performed in $O(|\mathcal{S}|^2|\mathcal{A}|)$. As defined earlier in Section 5.3, $|\mathcal{S}| = k \times l$ and $|\mathcal{A}| = l$ where k is the number of discrete environment states and l is the number of placement plans. Therefore, for solving TSTP using Value Iteration each iteration can be performed in $O(k^2 l^3)$.

In a trivial manner, the best case scenario for TSTP is to identify a single placement plan that is optimal for all environment states in the first phase, where a static task placement would suffice. However, a more realistic best case scenario is to acquire at most one optimal (or near-optimal) placement plan for each environment state in the first phase. In TSTP, the number of actions is equal to the number of placement plans.



(a) Latency measured for event generation with bursts of high event rates

(b) Latency measured for event generation in a bi-modal distribution

FIGURE 5.10: Summarised mean per-event latency (composed as the sum of processing and network latencies) for processing 10000 events of the ETL+STATS application in 5 static placement plans (Plan 4-6, 5-5, 6-3, 6-7, and RA), greedy placement plan and TSTP placement plan when the mean of the event generation rate is changing between 5evps, 10evps, 20evps, 50evps and 100evps. Each bar is annotated with the mean per-event latency value for each placement policy. Y-axis is in the symmetric log scale.

Therefore, based on the assumption that it is possible to identify a discrete number of environment states for the particular application, the running time of TSTP is bounded by the number of identified environment states instead of the number of all possible placement plans. For a large application with many tasks, this can reduce the state explosion significantly. However, the worst case for TSTP is to identify more than one optimal placement plan for each environment state. In this case, TSTP is bounded by the number of placement plans regardless of the number of discrete environment states, which results in state explosion for large scale applications. However, even in the worst case, by only choosing one optimal placement plan for each environment state in the first phase, it is possible to mitigate the state explosion problem in TSTP so that the running time is bounded by the number of environment states only. Selecting one optimal placement plan for each environment state can be conducted based on an application specific policy such as choosing the task placement that is deployed on a maximum number of resources. As a result, for TSTP, in the worst case, $l \leq k$ and consequently each iteration can be performed in $O(k^5)$ regardless of the number of possible placement plans. For example, despite the fact that in the ETL+STATS application there exists 42 possible placement plans, we only consider four plans which are identified as optimal for the five environment states in which we expect the application to operate. Therefore, by selecting appropriate environment states to represent an application scenario, TSTP can provide an optimal task migration policy that also mitigates the state explosion problem.

5.4.5 Deploying Multiple Applications with TSTP

Our problem formulation as well as the experiments, only consider a single application to be executed at a given time. However, it is reasonable to assume that multiple DSP applications may share the resource infrastructure. Here, we discuss how TSTP can be modified to handle this requirement.

When multiple applications are considered in the first phase, optimal placement plans need to be acquired for each application by first identifying the expected environment states and then considering the resource consumption, latency and power measurements of each individual application in each state. In the second phase, the reward function needs to consider the metrics related to each application such as considering the weighted sum of individual application latencies. The joined environment states of each application need to be considered by measuring how each application operates in each state and acquiring the relevant measurements to be used in the MDP rewards calculation. In addition, the transition probabilities need to consider the states related to each individual application, i.e., for some applications the probability of reaching certain configurations could be zero. Once the MDP model is formed, the TSTP policy can be acquired by solving the MDP followed by the same approach used for single-application execution. We leave conducting these experiments for future work.

5.5 Comparison with Related Work

Similar to the works of [159], [160], [161], and [162] listed in Section 2.5.1.3, in our study, we use a model-based MDP as for some DSP application scenarios, the environment conditions are known such that the states, reward function, and the transition probabilities between the states can be formed. In contrast to existing literature, our study introduces a novel methodology to reduce the state space to overcome the drawbacks of state explosion by utilising locally optimal placement plans only instead of all the possible task placement plans. Our study is the first of its kind to incorporate pre-deployed placement plans with semi-dynamic migration. In addition, we also introduce a configurable reward function for DSP applications. As per the taxonomy presented in Chapter 2, TSTP can be categorized as a semi-dynamic task placement strategy formed as a MDP with a multi-objective optimisation on edge and cloud computing infrastructure.

5.6 Conclusion

In this study, we presented a novel task placement strategy which models a DSP application executing on edge and cloud computing resources as a model-based Infinite Horizon Markov Decision Process with discounted rewards. In order to overcome the drawbacks of state explosion as well as the latency costs of fully dynamic task placement, we proposed the Two-phase Semi-dynamic Task Placement strategy (TSTP).

In the first phase of TSTP, we analysed the characteristics of the application and the environment. It is realistic to assume that we can analyse an existing DSP application to examine the expected operating conditions. For example, if we consider a fitness tracking or an augmented reality application, the lower and upper bounds for the event generation rate or event size can be identified (or even controlled by configuration) and then with reasonable accuracy the environment states could be arranged into discrete bins. Then we utilised a constraint based task placement strategy to acquire the locally optimal task placement for each environment state. In the second phase of TSTP, we formed our model-based MDP using the states, transition probabilities and the introduced configurable reward function. We then deployed the locally optimal task placement plans on to the underlying resources. By solving the MDP using the Value Iteration Algorithm we acquired an optimal stationary policy to switch between these already placed task placement plans in order to optimise the reward. We showed that when the environment conditions vary largely or when the application is complex, our TSTP policy was able to reduce the latency per event when compared with a greedy task placement as well as static task placements including the static resource-aware task placement. We also showed that, if discrete environment conditions could be identified, TSTP could be only bounded by the number of environment states in contrast to the number of possible placement plans, which can be substantially higher for large DSP applications and lead to state explosion.

Chapter 6

Conclusions and Future Directions

In-memory processing of unbounded, high velocity data streams to produce real-time, incremental results has been the main objective of the Data Stream Processing paradigm. Real-time processing becomes a critical requirement for multitude of stream processing applications such as augmented/virtual reality, federated learning, video processing and health monitoring applications. In addition, concerns for improving data privacy is emerging as another key requirement for these data stream processing applications. To cater to these requirements and to realise this paradigm, multiple generations of DSP frameworks have evolved which are often designed to be deployed on the distributed cloud computing infrastructure with a scalable, pay-as-you-go resource model. Bespoke DSP applications can be deployed on these frameworks where the framework handles ingesting streams arriving from an event source to the application, conducting the processing based on the application logic, and transferring the resulting streams to the destinations as configured, along with the auxiliary operations such as measuring performance or failure handling. A typical application is formed as a connected graph of decoupled tasks known as sources, operators and sinks, each connected with a data flow. To execute an application, the tasks need to be placed onto computing resources, which is referred to as the task placement, and connected with the downstream tasks over the communication media between the computing resources, based on the topology of the graph.

With the widespread adaptation of connected devices such as Internet of Things, many sources that generate unbounded event streams have moved outside the cloud. For latency critical applications that rely on these event streams, the communication overhead between the devices and cloud infrastructure becomes a detrimental factor where

the conventional approach of processing on the cloud becomes infeasible. Due to the rapid growth of these devices, edge computing has emerged as a complementing programming paradigm to cloud computing. The edge devices are often either the event sources themselves or they can be located closer to the event sources. With the advancements of microprocessor and networking technologies, these devices present a potential resource infrastructure for sharing the workload of data stream processing with the cloud. Since many of these devices are owned by the end users, processing and storing data on them mitigates privacy concerns as well. However, due to the concerns such as the limitations on the resource availability, the heterogeneity of the device configurations, the concerns for power consumption, the overhead caused by wireless networks, and the volatility of the devices, a trivial workload sharing strategy is proven to be inefficient. It is our expectation that the models and solutions that were presented here, will expand the methodologies of utilising edge and cloud resources for stream processing, and improve the use of these methodologies for practical applications.

We conclude the thesis by presenting our key ideas in Section 6.1 where we first summarise our understanding of the prior work that we build our study upon. Next, we discuss the model-based, open-source, discrete event simulator for running experiments to realise our goals, followed by the static task placement and optimisation framework which produce optimal task placement plans for DSP applications that largely operate under static conditions and conclude with the semi-dynamic task placement framework for applications that operate under dynamic conditions. Finally, we describe a set of future directions in Section 6.2 where we discuss about further challenges that need to be overcome to improve the quality of service provided to end users with the help of edge computing for data stream processing.

6.1 Summary and Conclusions

In Chapter 1, we introduced the concepts of Data Stream Processing, Edge Computing and Cloud Computing which are the foundations of this study. Then in Chapter 2, we reviewed the relevant literature in order to develop a taxonomy that was in turn used to identify the gaps that need to be addressed. Here, we have identified that the existing work can be classified into four main categories; 1) Application – where the characteristics of DSP applications such as their composition, service delivery mechanism are defined, 2) Infrastructure – where the computing resources such as the cloud and edge resources are discussed, 3) Task placement – where the aspects of task placement such as the modelling strategy, optimisation objectives, and dynamism of the placement proposed by prior work are analysed, and 4) Evaluation tools – where the tools that are

being used for evaluating the performance of proposed solutions are discussed. Based on our review of the prior work, we identified that an extendible model that defines the characteristics of DSP applications executed on shared edge and cloud infrastructure is lacking and consequently the need for a simulation environment based on such a model can be a vital contribution as an evaluation tool. We also identified that a task placement strategy that identifies the resource and residency constraints of these applications is absent in the state-of-the-art for applications that operate under static and dynamic conditions. In the following Sections 6.1.1 to 6.1.3, we summarise our contributions that aim to overcome these challenges.

6.1.1 A Model-based Simulator for Distributed Data Stream Processing

In this thesis, we first introduced ECSNeT++, a novel simulator that was implemented using our model of data stream processing applications running on edge and cloud computing infrastructure. This model represented the characteristics of DSP applications such as the event generation rate and event size distributions of sources, selectivity and productivity ratio distributions of operators, task placement and routing of data flow between the tasks, and measurement of latency and power consumption of the overall application with fine granularity. We also modelled the processing characteristics of a multi-core, multi-threaded processor from the perspective of a DSP application, a state based power consumption model, and the ability to set-up heterogeneous distributed network configurations that utilise these components. Based on this model, we implemented our open-source and publicly available discrete event simulation toolkit, which was built on the widely used OMNeT++ network simulator.

Our next focus was on calibrating and verifying the simulator for measuring the accuracy of the recorded metrics against a real device test-bed. Here, we first set-up a heterogeneous edge and cloud computing test-bed with WLAN and LAN network that connects the devices, and created an identical simulated environment in the ECSNeT++. We then calibrated ECSNeT++ with the networking and power consumption parameters of the real test-bed. By deploying real world applications with multiple task placement plans on both set-ups, we showed that ECSNeT++ can acquire comparable estimations of latency and power consumption measurements.

However, there were some limitations that needed to be addressed. First of all, the lack of variation in simulated network delay measurements of some plans remained an issue that can be improved. The variations in the network delay values in experiments run on real test-bed were caused by the variations in the transmission latency which in turn was largely caused by the intricacies of the network such as packet retransmission,

collision avoidance and packet loss. A more detailed network modelling would be desired in scenarios where it is difficult to isolate the network between the edge and the cloud.

In the real experiments illustrated in Section 3.4, the processing model showed higher inter-quartile range in each scenario as expected, due to the variations that can happen with the overheads in the underlying CPU architecture such as context switching or interruptions. Since we were using a fixed per-event processing delay for each task, a similar variability was absent in the simulated results.

When it came to the power and energy consumption, we found that in our simulation toolkit, the idle energy consumption to be the largest contributor for power consumption which was the base power consumption as expected. However, since it was not possible to measure the individual power consumption components for each event (processing power, transmission power, and idle power) in our real experimental set-up, we were unable to properly compare the distribution of components of the power consumption between the simulation and the real set-up. In addition, the state based power consumption model that we implemented required careful calibration of power consumption values associated with each state change, in order for it to be accurate. Therefore, while ECSNeT++ can be used to gather preliminary data on energy consumption, a more detailed power and energy consumption model would be a valuable extension to ECSNeT++.

As a final observation, it is important to note that using proper calibration parameters for ECSNeT++ with respect to a real experimental set-up, was the key step in acquiring reliable measurements for the conducted simulated experiments.

Further improvements can be done to the modelling of network transfer time and event processing time by incorporating an external and internal queueing model, as illustrated by Cooper, Ezhilchelvan and Mitrani [262], which closely represents internal architecture of some of the existing data stream processing engines.

6.1.2 An Optimisation Framework for Task Placement for Applications Operating Under Static Conditions

In order to share the workload of DSP applications with the edge and cloud computing resources, an optimal task placement strategy was required. The task placement needed to consider the resource requirements, residency requirements, and service requirements of the application, and then it needed to consider if the available resources could satisfy these requirements. Finally, the most optimal task placement from the available task placements required to be identified. To achieve this, we modelled the task placement problem as a constraint satisfaction and optimisation problem and acquired optimal (or

near-optimal) task placement plans as solutions to the problem. We added residency constraints to specify the residency requirements of applications such as requiring to place a user-specific data processing tasks on devices owned by the user. The resource constraints of our model were used to ensure that the resource requirements such as CPU or power were satisfied by the proposed task placements. Our model introduced a region based design to identify the edge and cloud resources and to collocate tasks that are designed specifically to be executed on these devices. We also used the sum of individual latencies as the optimisation objective in order to show that by minimising that we can minimise the overall end-to-end latency even in multi-path topologies. Our experiments run using ECSNeT++ showed that the proposed static optimisation framework results in $8 - 14\%$ reduction of mean end-to-end latency, and $\approx 14 - 15\%$ total per-event energy consumption improvement against the conventional cloud only placement, and $\approx 14 - 16\%$ reduction in mean end-to-end latency and $\approx 1 - 5\%$ reduction in total per-event energy consumption when compared with a naive edge only task placement. We also showed that our solution achieves the latency optimal task placement plan comparable to a solution acquired through exhaustive brute force.

Our main assumption on the applications considered in this study was that the operating conditions such as the event generation rate or the size of the generated events remain fixed during the lifetime of the application such as video processing applications that process a constant feed of video frames. The proposed optimised static task placement framework can be effectively used to satisfy the requirements of such applications instead of relying on conventional cloud only task placement or exhaustively deriving an optimal task placement which becomes intractable for large scale applications.

6.1.3 A Semi-Dynamic Task Placement Strategy for Applications Operating Under Dynamic Conditions

In many real world applications, the operating conditions change with time. For example, applications such as a fitness tracking or taxi route prediction that rely on the human behaviour tend to display bimodal distributions in their event generation rates. When the operating conditions change, the task placement may also need to change in order to provide the optimised quality of service to the end users. However, for real time applications, constant switching between multiple task placement plans can prove to be infeasible as it has a detrimental effect on the end-to-end latency due to the overhead associated with the migration. We presented a semi-dynamic task placement strategy in order to overcome this challenge in our final work of this thesis.

Our strategy was two phased. In the first phase, we analysed the application to identify the distribution of the changing conditions where we aimed to generate discrete bins by identifying the upper and lower bounds of the distribution. Once the discrete bins of expected operating conditions were identified, which we called as environment states, we utilised the static task placement framework to acquire one optimal task placement for each discrete state. In our analysis of the application, we also aimed to acquire the associated transition probabilities between the discrete states. With this information, in the second the phase we formed a model-based, infinite horizon, discounted reward, Markov decision process with the environment states, locally optimal task placement plans, and transition probabilities. The actions of the process were defined as the action of selecting a task placement plan. We also introduced a configurable reward function that returns higher rewards for switching to an optimal task placement plan or staying on an optimal task placement plan given the current and next environment states. Then we pre-deployed the identified optimal task placement plans onto the edge and cloud computing resources using our ECSNeT++ simulator. Here, the simulator was designed to switch between the pre-deployed plans based on a policy. The optimal stationary policy for switching between deployed plans, TSTP policy, was acquired by solving the Markov Decision Process using the Value Iteration algorithm. With this strategy we showed that we could acquire optimal latency results when compared with static task placements, resource aware task placements as well as fully dynamic greedy task placement. In synthetic and real world applications, we were able to improve the latency by 10.45% - 17.41% on average, when compared to the greedy approach. We also showed that based on the assumption that a finite number of discrete environment states are identified, TSTP was able to mitigate the state explosion problem that renders MDP solutions intractable by bounding the application runtime complexity to the number of discrete environment states instead of the number of all possible task placement plans.

6.2 Future Directions

With our work as described throughout this thesis, and with the familiarity of the prior work, we believe exploring the following future directions would be beneficial for further establishing our objective to utilise edge computing resources for data stream processing.

6.2.1 Optimising Task Placement for Multi-tenant DSP Applications

Multi-tenant applications share the resources among other users and applications. Often, virtualisation is used to ensure the isolation between the environments which are owned by individual tenants where multiple applications owned by different users may run on

the same resource in dedicated containers. Multi-tenant applications are widely used in the cloud computing domain while they are gaining traction in some edge based studies such as edge and cloud computing use cases in smart cities. For instance, Morabito et al. [263] explore the possibility of lightweight virtualisation techniques to facilitate multi-tenancy on IoT based edge computing resources by using containerisation or unikernels. When task placement decisions are made for multi-tenant applications, we need to consider the individual requirements of each application as well as their interactions such as the deployment of static and dynamic applications by different tenants. Therefore, the task placement strategies need to be aware of the fluctuations in the resource demands by multi-tenant applications.

6.2.2 Extending Two-phase Semi-dynamic Task Placement as a Model-free Markov Decision Process

Our two-phase semi-dynamic task placement strategy was based on the assumption that the operating conditions of the application can be analysed prior to deployment where the discrete environment states and their transition probabilities were known. Based on this knowledge, we were able to form a model-based Markov Decision Process which produced an optimal stationary policy for switching between task placement plans. However, for some stream processing applications, identifying the transition probabilities between the states or the reward associated with each action may prove to be challenging if no prior knowledge of the application is available. For these applications, a model-free Markov Decision Process may become a viable solution. Here, a reinforcement learning approach can be utilised to learn the model by using trial and error to identify the rewards associated with task placements under varying conditions. This approach can lead to multiple research pathways where diverse learning techniques can be utilised for forming the models and consequently devising optimal task placement policies based on the learned models.

6.2.3 Handling Fault Tolerance

Some edge devices are volatile and prone to failures. As a result, considering the probability of device failures when optimising for task placement is required for robust execution of the DSP applications. One of the most prominent approaches for enabling fault tolerance in DSP applications is using checkpoints. Here, depending on the granularity of the checkpointing, the application level states, task level states or individual device level states can be stored periodically in a reliable storage mechanism to be re-used in the presence of failures. Multiple challenges need to be overcome to implement fault tolerance

with checkpointing-enabled task placement for DSP applications. The check-pointing frequency can have a detrimental effect on the end-to-end latency, if the application execution needs to be paused for taking snapshots. Therefore, the checkpointing frequency needs to be determined based on the factors such as the checkpoint cost and expected failure rate [264], all of which need to be considered as a latency trade-off in the task placement optimisation problem. Efficient heuristics are also needed to determine how to conduct device level checkpointing for collocated tasks. In addition, the storage availability for storing checkpoints, which can be a challenging factor in resource constrained edge devices, needs to be considered in the optimisation problem as well.

The proposed two-phase semi-dynamic task placement strategy for applications with dynamic operating conditions, can be configured using the availability of the resources as another environment state where if some resources become unavailable, the locally optimal task placement for the available resources can be switched on. The transition probabilities for these states can be determined based on the failure distribution. However, transition cost associated with task switching needs to be carefully determined in order to make sure the effect on end-to-end latency as well as other relevant metrics are within guaranteed ranges.

Similar to failure handling, failure detection can also play a vital role in edge networks. As these networks tend to be formed using numerous distributed devices, a higher communication overhead may be inflicted if a failure detection mechanism such as sending periodic heart beats is utilised [50]. Consequently, the communication overhead can have a detrimental effect on the real-time applications. Therefore, the frequency of such failure detection mechanisms needs to be taken into account in the latency optimisation problem for task placement.

6.2.4 Implementing Support for Privacy and Security Policies

Privacy and security have been concerns for cloud computing where the users are concerned about where data is sent, processed and stored. An advantage of edge computing, unlike cloud computing, is the ability to execute applications on devices owned by users [34]. As a result, with the decoupled tasks in a DSP application, a user may decide to run a privacy sensitive task on a computing resource owned by the user, such as processing keyboard inputs of a user on the user's mobile device itself for next-word prediction. The residency constraints of the presented constraint satisfaction model support adding privacy related residency constraints at a crude level. The works such as AND-OR security requirements trees presented by Russo Russo et al. [265] and the secure task placement methodology by considering specific application requirements presented by Forti, Ferrari

and Brogi [266] can be extended for implementing security and privacy aware task placement. Such a framework that supports implementing comprehensive and fine-grained, security and privacy policies for task placement problems, is a valuable improvement that can be addressed in the future.

6.3 Final Remarks

Our objective of this study was to overcome the challenges of sharing the workload of stream processing applications between edge and cloud computing resources and to demonstrate that it can be beneficial for the real time stream processing applications that rely on the event streams generated by multitude of smart, connected, edge devices that are increasingly becoming available.

To analyse the behaviour of these real time stream processing applications, a comprehensive model of the application as well as the resources was required and to analyse the performance of the applications running on these resources, experimental test-beds were necessary. We showed that ECSNeT++ can mitigate the need for such a real test-bed and allow to run numerous experiments with varying conditions to acquire accurate estimations of the application latency as well as the power consumption metrics, given that ECSNeT++ is properly calibrated. With the open source and public availability and extendibility of ECSNeT++, it can be further enhanced and used as a practical toolkit for acquiring preliminary results. The static and semi-dynamic task placement frameworks proposed in this study are able to capture the requirements of applications that operate under fixed or varying conditions and provide a general approach to acquire optimal task placement policies that reduce the end-to-end latency as well as the power consumption requirements of these applications. We believe, the research outcome of this thesis is instrumental in devising effective and practical utilisation strategies for executing distributed data stream processing applications on edge and cloud computing infrastructure.

Bibliography

[1] Anshu Shukla, Shilpa Chaturvedi and Yogesh Simmhan. 'RIoTBench: A Real-time IoT Benchmark for Distributed Stream Processing Platforms'. In: (2017). DOI: 10.5120/19787-1571. arXiv: 1701.08530.

[2] *Apache Flink: Stateful Computations over Data Streams*. https://flink.apache.org/. (Accessed on 11/26/2020).

[3] *Apache Beam*. https://beam.apache.org/. (Accessed on 11/26/2020).

[4] *Apache Storm*. https://storm.apache.org/. (Accessed on 11/26/2020).

[5] *Apache Heron · A realtime, distributed, fault-tolerant stream processing engine*. https://heron.incubator.apache.org/. (Accessed on 11/26/2020).

[6] *Edge TPU - Run Inference at the Edge — Edge TPU — Google Cloud*. https://cloud.google.com/edge-tpu/. (Accessed on 05/2019).

[7] *AWS IoT Greengrass - Amazon Web Services*. https://aws.amazon.com/greengrass/. (Accessed on 03/09/2020).

[8] *Microsoft Azure IoT Edge*. https://azure.microsoft.com/en-au/services/iot-edge/. (Accessed on 03/09/2020).

[9] *Google Nest Cam*. https://nest.com/cameras/. (Accessed on 11/2018).

[10] *Piper: All-in-one wireless security system*. https://getpiper.com/howitworks/. (Accessed on 11/2018).

[11] *ProVigil Target Tracking and Analysis*. https://pro-vigil.com/features/video-analytics/object-tracking/. (Accessed on 11/2018).

[12] *Connode Smart Metering*. https://cyanconnode.com/. (Accessed on 11/2018).

[13] *Dust Networks Applications: Industrial Automation*. https://www.analog.com/en/applications/\technology/smartmesh-pavilion-home/dust-networks-applications.html. (Accessed on 11/2018).

[14] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon and Daniel Ramage. 'Federated Learning for Mobile Keyboard Prediction'. In: (2018). arXiv: 1811.03604.

[15] Geetika T. Lakshmanan, Ying Li and Rob Strom. 'Placement strategies for internet-scale data stream systems'. In: *IEEE Internet Computing* 12.6 (2008), pp. 50–60. DOI: 10.1109/MIC.2008.129.

[16] Raphael Eidenbenz and Thomas Locher. 'Task Allocation for Distributed Stream Processing'. In: *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*. 2016, pp. 1–9. DOI: 10.1109/INFOCOM.2016.7524433. arXiv: 1601.06060.

[17] Ying Xing, Stan Zdonik and Jeong Hyon Hwang. 'Dynamic load distribution in the borealis stream processor'. In: *Proceedings - International Conference on Data Engineering*. Icde. 2005, pp. 791–802. DOI: 10.1109/ICDE.2005.53.

[18] Boyang Peng, Mohammad Hosseini, Zhihao Hong, Reza Farivar and Roy Campbell. 'R-Storm: Resource-Aware Scheduling in Storm'. In: *Proceedings of the 16th Annual Middleware Conference on - Middleware '15*. 2015, pp. 149–161. DOI: 10.1145/2814576.2814808.

[19] Rajrup Ghosh and Yogesh Simmhan. 'Distributed Scheduling of Event Analytics across Edge and Cloud'. In: *CoRR* (2016), pp. 1–31. arXiv: 1608.01537.

[20] Valeria Cardellini, Vincenzo Grassi, Francesco Lo Presti and Matteo Nardelli. 'Optimal operator placement for distributed stream processing applications'. In: *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems*. 2016, pp. 69–80. DOI: 10.1145/2933267.2933312.

[21] Valeria Cardellini, Vincenzo Grassi, Francesco Lo Presti and Matteo Nardelli. 'Optimal Operator Replication and Placement for Distributed Stream Processing Systems'. In: *ACM SIGMETRICS Performance Evaluation Review*. Vol. 44. 4. 2017, pp. 11–22. DOI: 10.1145/3092819.3092823.

[22] Gene M. Amdahl. 'Validity of the single processor approach to achieving large scale computing capabilities'. In: *AFIPS Conference Proceedings - 1967 Spring Joint Computer Conference, AFIPS 1967* (1967), pp. 483–485. DOI: 10.1145/1465482.1465560.

[23] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg and Ivona Brandic. 'Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility'. In: *Future Generation Computer Systems* 25.6 (2009), pp. 599–616. DOI: 10.1016/j.future.2008.12.001.

[24] *Gartner Forecasts Worldwide Public Cloud Revenue to Grow 6.3% in 2020*. https://www.gartner.com/en/newsroom/press-releases/2020-07-23-gartner-forecasts-worldwide-public-cloud-revenue-to-grow-6point3-percent-in-2020. (Accessed on 08/07/2020).

[25] Asthon. Kevin. 'That ' Internet of Things ' Thing'. In: *RFiD Journal* (2010), p. 4986. DOI: 10.1038/nature03475.

[26] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres and Nigel Davies. 'The case for vm-based cloudlets in mobile computing'. In: *IEEE pervasive Computing* 8.4 (2009), pp. 14–23.

[27] Mahadev Satyanarayanan. 'The emergence of edge computing'. In: *Computer* 50.1 (2017), pp. 30–39.

[28] China Mobile. 'C-RAN: the road towards green RAN'. In: *White paper, ver* 2 (2011), pp. 1–10.

[29] Zhenbo Zhu, Parul Gupta, Qing Wang, Shivkumar Kalyanaraman, Yonghua Lin and Hubertus Franke. 'Virtual Base Station Pool : Towards A Wireless Network Cloud for Radio Access Networks Categories and Subject Descriptors'. In: *Proc. 8th ACM International Conference on Computing Frontiers* (2011).

[30] Karthik Kumar and Yung-Hsiang Lu. 'Cloud computing for mobile users: Can offloading computation save energy?' In: *Computer* 43.4 (2010), pp. 51–56.

[31] Flavio Bonomi, Rodolfo Milito, Jiang Zhu and Sateesh Addepalli. 'Fog computing and its role in the internet of things'. In: *Proceedings of the first edition of the MCC workshop on Mobile cloud computing - MCC '12*. 2012, p. 13. DOI: 10.1145/2342509.2342513. arXiv: arXiv:1502.01815v3.

[32] Amir Vahid Dastjerdi, H. Gupta, R. N. Calheiros, S. K. Ghosh and Rajkumar Buyya. 'Fog Computing: Principles, architectures, and applications'. In: *Internet of Things: Principles and Paradigms* (2016), pp. 61–75. DOI: 10.1016/B978-0-12-805395-9.00004-6. arXiv: 1601.02752.

[33] Sophia Antipolis. *ETSI - ETSI announces first meeting of new standardization group on Mobile-edge Computing*. https://www.etsi.org/newsroom/news/838-2014-10-news-etsi-announces-first-meeting-of-new-standardization-group-on-mobile-edge-computing. (Accessed on 08/08/2020).

[34] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li and Lanyu Xu. 'Edge Computing: Vision and Challenges'. In: *IEEE Internet of Things Journal* 3.5 (2016), pp. 637–646. DOI: 10.1109/JIOT.2016.2579198.

[35] Yongli Zhao, Wei Wang, Yajie Li, Carlos Colman Meixner, Massimo Tornatore and Jie Zhang. 'Edge Computing and Networking: A Survey on Infrastructures and Applications'. In: *IEEE Access* 7 (2019), pp. 101213–101230. DOI: 10.1109/access.2019.2927538.

[36] Michael Till Beck, Martin Werner, Sebastian Feld and Thomas Schimper. 'Mobile Edge Computing : A Taxonomy'. In: *Proc. of the Sixth International Conference on Advances in Future Internet.* c (2014), pp. 48–54.

[37] Congfeng Jiang, Xiaolan Cheng, Honghao Gao, Xin Zhou and Jian Wan. 'Toward Computation Offloading in Edge Computing: A Survey'. In: *IEEE Access* 7 (2019), pp. 131543–131558. DOI: 10.1109/ACCESS.2019.2938660.

[38] Blesson Varghese, Nan Wang, Sakil Barbhuiya, Peter Kilpatrick and Dimitrios S. Nikolopoulos. 'Challenges and Opportunities in Edge Computing'. In: *Proceedings - 2016 IEEE International Conference on Smart Cloud, SmartCloud 2016* (2016), pp. 20–26. DOI: 10.1109/SmartCloud.2016.18. arXiv: 1609.01967.

[39] Wazir Zada Khan, Ejaz Ahmed, Saqib Hakak, Ibrar Yaqoob and Arif Ahmed. 'Edge computing: A survey'. In: *Future Generation Computer Systems* 97 (2019), pp. 219–235. DOI: 10.1016/j.future.2019.02.050.

[40] Shanhe Yi, Zijiang Hao, Zhengrui Qin and Qun Li. 'Fog computing: Platform and applications'. In: *Proceedings - 3rd Workshop on Hot Topics in Web Systems and Technologies, HotWeb 2015*. 2016, pp. 73–78. DOI: 10.1109/HotWeb.2015.22. arXiv: arXiv:1502.01815v3.

[41] Carlo Puliafito, Enzo Mingozzi, Francesco Longo, Antonio Puliafito and Omer Rana. 'Fog computing for the Internet of Things: A survey'. In: *ACM Transactions on Internet Technology* 19.2 (2019). DOI: 10.1145/3301443.

[42] Hamid Reza Arkian, Abolfazl Diyanat and Atefe Pourkhalili. 'MIST: Fog-based data analytics scheme with cost-efficient resource provisioning for IoT crowdsensing applications'. In: *Journal of Network and Computer Applications* 82.December 2016 (2017), pp. 152–165. DOI: 10.1016/j.jnca.2017.01.012.

[43] Marcos Dias de Assunção, Alexandre da Silva Veith and Rajkumar Buyya. 'Distributed data stream processing and edge computing: A survey on resource elasticity and future directions'. In: *Journal of Network and Computer Applications* 103 (2018), pp. 1–17.

[44] Jianjun Chen, David J. DeWitt, Feng Tian and Yuan Wang. 'NiagaraCQ: A scalable continuous query system for internet databases'. In: *SIGMOD Record (ACM Special Interest Group on Management of Data)* 29.2 (2000), pp. 379–390. DOI: 10.1145/335191.335432.

[45] Arvind Arasu et al. 'STREAM: The Stanford Stream Data Manager (Demonstration Description)'. In: *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data* 26.March 2003 (2003), pp. 19–26. DOI: 10.1145/872757.872854.

[46] The STREAM Group. *STREAM: The Stanford Stream Data Manager*. Technical Report 2003-21. Stanford InfoLab, 2003.

[47] Daniel J Abadi, Don Carney, Ugur Etintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul and Stan Zdonik. 'Aurora: a new model and architecture for data stream management'. In: *The VLDB Journal The International Journal on Very Large Data Bases* 12.2 (2003), pp. 120–139. DOI: 10.1007/s00778-003-0095-z.

[48] Magdalena Balazinska, Hari Balakrishnan and Mike Stonebraker. 'Contract-based load management in federated distributed systems'. In: *Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation*. 2004, pp. 1–14.

[49] Daniel J Abadi et al. 'The Design of the Borealis Stream Processing Engine.' In: *{CIDR} 2005, Second Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4-7, 2005, Online Proceedings*. www.cidrdb.org, 2005, pp. 277–289. DOI: http://www.cidrdb.org/cidr2005/papers/P23.pdf.

[50] Ankit Toshniwal et al. 'Storm@twitter'. In: *Proceedings of the 2014 ACM SIGMOD international conference on Management of data - SIGMOD '14* (2014), pp. 147–156. DOI: 10.1145/2588555.2595641.

[51] Sanjeev Kulkarni, Nikunj Bhagat, Masong Fu, Vikas Kedigehalli, Christopher Kellogg, Sailesh Mittal, Jignesh M. Patel, Karthik Ramasamy and Siddarth Taneja. 'Twitter Heron'. In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data - SIGMOD '15* (2015), pp. 239–250. DOI: 10.1145/2723372.2742788.

[52] Paris Carbone, Stephan Ewen, Seif Haridi, Asterios Katsifodimos, Volker Markl and Kostas Tzoumas. 'Apache Flink: Unified Stream and Batch Processing in a Single Engine'. In: *Data Engineering* (2015), pp. 28–38.

[53] Tyler Akidau et al. 'The dataflow model'. In: *Proceedings of the VLDB Endowment* 8.12 (2015), pp. 1792–1803. DOI: 10.14778/2824032.2824076.

[54] *Apache Samza*. http://samza.apache.org/. (Accessed on 08/06/2020).

[55] Hooman Peiro Sajjad, Ken Danniswara, Ahmad Al-Shishtawy and Vladimir Vlassov. 'SpanEdge: Towards unifying stream processing over central and near-the-edge data centers'. In: *Proceedings - 1st IEEE/ACM Symposium on Edge Computing, SEC 2016* (2016), pp. 168–178. DOI: 10.1109/SEC.2016.17.

[56] Xinwei Fu, Talha Ghaffar, James C. Davis and Dongyoon Lee. 'Edgewise: A better stream processing engine for the edge'. In: *Proceedings of the 2019 USENIX Annual Technical Conference, USENIX ATC 2019* (2019), pp. 929–945.

[57] *Apache Edgent*. https://edgent.incubator.apache.org/. (Accessed on 08/06/2020).

[58] Farah Ait Salaht, Frédéric Desprez and Adrien Lebre. 'An overview of service placement problem in Fog and Edge Computing'. In: *ACM Computing Surveys (CSUR)* 53.3 (2020), pp. 1–35.

[59] S M Riazul Islam, Daehan Kwak, Humaun Kabir, Mahmud Hossain and Kyung-Sup Kwak. 'The Internet of Things for Health Care : A Comprehensive Survey'. In: *Access, IEEE* 3 (2015), pp. 678 –708. DOI: 10.1109/ACCESS.2015.2437951.

[60] Zeyi Tao and College William. 'eSGD : Communication Efficient Distributed Deep Learning on the Edge'. In: (), pp. 1–6.

[61] G Premsankar, M Di Francesco and T Taleb. 'Edge Computing for the Internet of Things: A Case Study'. In: *IEEE Internet of Things Journal* PP.99 (2018), p. 1. DOI: 10.1109/JIOT.2018.2805263.

[62] Rodrigo Izidoro Tinini, Larissa C.M. Reis, Daniel Mac Edo Batista, Gustavo Bittencourt Figueiredo, Massimo Tornatore and Biswanath Mukherjee. 'Optimal Placement of Virtualized BBU Processing in Hybrid Cloud-Fog RAN over TWDM-PON'. In: *2017 IEEE Global Communications Conference, GLOBECOM 2017 - Proceedings* 2018-January (2017), pp. 1–6. DOI: 10.1109/GLOCOM.2017.8254770.

[63] Redowan Mahmud, Satish Narayana Srirama, Kotagiri Ramamohanarao and Rajkumar Buyya. 'Quality of Experience (QoE)-aware placement of applications in Fog computing environments'. In: *Journal of Parallel and Distributed Computing* 132 (2019), pp. 190–203. DOI: 10.1016/j.jpdc.2018.03.004.

[64] Haisheng Tan, Zhenhua Han, Xiang Yang Li and Francis C.M. Lau. 'Online job dispatching and scheduling in edge-clouds'. In: *Proceedings - IEEE INFOCOM* (2017). DOI: 10.1109/INFOCOM.2017.8057116.

[65] V. B. Souza, X. Masip-Bruin, E. Marín-Tordera, S. Sànchez-López, J. Garcia, G. J. Ren, A. Jukan and A. Juan Ferrer. 'Towards a proper service placement in combined Fog-to-Cloud (F2C) architectures'. In: *Future Generation Computer Systems* 87.2018 (2018), pp. 1–15. DOI: 10.1016/j.future.2018.04.042.

[66] Enrique Saurez, Kirak Hong, Dave Lillethun, Umakishore Ramachandran and Beate Ottenwälder. 'Incremental deployment and migration of geo-distributed situation awareness applications in the fog'. In: *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems - DEBS '16* (2016), pp. 258–269. DOI: 10.1145/2933267.2933317.

[67] José Santos, Tim Wauters, Bruno Volckaert and Filip De Turck. 'Resource provisioning for IoT application services in smart cities'. In: *2017 13th International Conference on Network and Service Management, CNSM 2017* 2018-January (2017), pp. 1–9. DOI: 10.23919/CNSM.2017.8255974.

[68] Quang Tran Minh, Duy Tai Nguyen, An Van Le, Hai Duc Nguyen and Anh Truong. 'Toward service placement on fog computing landscape'. In: *2017 4th NAFOSTED Conference on Information and Computer Science, NICS 2017 - Proceedings* 2017-January (2017), pp. 291–296. DOI: 10.1109/NAFOSTED.2017.8108080.

[69] Olena Skarlat, Matteo Nardelli, Stefan Schulte and Schahram Dustdar. 'Towards QoS-Aware Fog Service Placement'. In: *Proceedings - 2017 IEEE 1st International Conference on Fog and Edge Computing, ICFEC 2017*. 2017, pp. 89–96. DOI: 10.1109/ICFEC.2017.12.

[70] Deze Zeng, Lin Gu and Hong Yao. 'Towards energy efficient service composition in green energy powered Cyber–Physical Fog Systems'. In: *Future Generation Computer Systems* 105 (2020), pp. 757–765. DOI: 10.1016/j.future.2018.01.060.

[71] Marek Rychlý, Petr Škoda and Pavel Šmrž. 'Scheduling decisions in stream processing on heterogeneous clusters'. In: *Proceedings - 2014 8th International Conference on Complex, Intelligent and Software Intensive Systems, CISIS 2014*. 2014, pp. 614–619. DOI: 10.1109/CISIS.2014.94.

[72] Bin Cheng, Apostolos Papageorgiou and Martin Bauer. 'Geelytics: Enabling on-demand edge analytics over scoped data sources'. In: *Proceedings - 2016 IEEE International Congress on Big Data, BigData Congress 2016* (2016), pp. 101–108. DOI: 10.1109/BigDataCongress.2016.21.

[73] Valeria Cardellini, Francesco Lo Presti, Matteo Nardelli and Gabriele Russo Russo. 'Optimal operator deployment and replication for elastic distributed data stream processing'. In: *Concurrency and Computation: Practice and Experience* June (2017), e4334. DOI: 10.1002/cpe.4334.

[74] Beate Ottenwälder, Boris Koldehofe, Kurt Rothermel and Umakishore Ramachandran. 'MigCEP: Operator migration for mobility driven distributed complex event processing'. In: *DEBS 2013 - Proceedings of the 7th ACM International Conference on Distributed Event-Based Systems* (2013), pp. 183–194. DOI: 10.1145/2488222.2488265.

[75] Isaac Lera, Carlos Guerrero and Carlos Juiz. 'Availability-aware service placement policy in fog computing based on graph partitions'. In: *IEEE Internet of Things Journal* 6.2 (2019), pp. 3641–3651. DOI: 10.1109/JIOT.2018.2889511.

[76] Vincenzo De Maio and Ivona Brandic. 'First hop mobile offloading of DAG computations'. In: *Proceedings - 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID 2018* (2018), pp. 83–92. DOI: 10.1109/CCGRID.2018.00023.

[77] Marc Barcelo, Alejandro Correa, Jaime Llorca, Antonia M. Tulino, Jose Lopez Vicario and Antoni Morell. 'IoT-Cloud Service Optimization in Next Generation Smart Environments'. In: *IEEE Journal on Selected Areas in Communications* 34.12 (2016), pp. 4077–4090. DOI: 10.1109/JSAC.2016.2621398.

[78] Bruno Donassolo, Ilhem Fajjari, Arnaud Legrand and Panayotis Mertikopoulos. 'Fog Based Framework for IoT Service Orchestration'. In: *2019 16th IEEE Annual Consumer Communications and Networking Conference, CCNC 2019* (2019), pp. 5–10. DOI: 10.1109/CCNC.2019.8651852.

[79] Julien Gedeon, Michael Stein, Jeff Krisztinkovics, Patrick Felka, Katharina Keller, Christian Meurisch, Lin Wang and M Max. 'From Cell Towers to Smart Street Lamps : Placing Cloudlets on Existing Urban Infrastructures'. In: (2018). DOI: 10.1109/SEC.2018.00021.

[80] Rajrup Ghosh, Siva Prakash Reddy Komma and Yogesh Simmhan. 'Adaptive energy-aware scheduling of dynamic event analytics across edge and cloud resources'. In: *Proceedings - 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID 2018* (2018), pp. 72–82. DOI: 10.1109/CCGRID.2018.00022. arXiv: 1801.01087.

[81] Mohit Taneja and Alan Davy. 'Resource Aware Placement of IoT Application Modules in Fog-Cloud Computing Paradigm'. In: *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)* (2017), pp. 1222–1228. DOI: 10.23919/INM.2017.7987464.

[82] Redowan Mahmud, Satish Narayana Srirama, Kotagiri Ramamohanarao and Rajkumar Buyya. 'Quality of Experience (QoE) -aware Placement of Applications in Fog Computing Environments'. In: *J. Parallel Distrib. Comput.* April (2018). DOI: 10.1016/j.jpdc.2018.03.004.

[83] Bongjun Kim, Seonyeong Heo, Gyeongmin Lee, Seungbin Song, Jong Kim and Hanjun Kim. 'Spinal code: Automatic code extraction for near-user computation in fogs'. In: *ACM International Conference Proceeding Series* (2019), pp. 87–98. DOI: 10.1145/3302516.3307356.

[84] Jakob Mass, Chii Chang and Satish Narayana Srirama. 'Context-aware edge process management for mobile thing-to-fog environment'. In: *ACM International Conference Proceeding Series* (2018). DOI: 10.1145/3241403.3241449.

[85] Shreshth Tuli, Redowan Mahmud, Shikhar Tuli and Rajkumar Buyya. 'FogBus: A Blockchain-based Lightweight Framework for Edge and Fog Computing'. In: *Journal of Systems and Software* 154 (2019), pp. 22–36. DOI: 10.1016/j.jss.2019.04.050. arXiv: 1811.11978.

[86] Jose Santos, Tim Wauters, Bruno Volckaert and Filip De Turck. 'Towards network-Aware resource provisioning in kubernetes for fog computing applications'. In: *Proceedings of the 2019 IEEE Conference on Network Softwarization: Unleashing the Power of Network Softwarization, NetSoft 2019* (2019), pp. 351–359. DOI: 10.1109/NETSOFT.2019.8806671.

[87] Shweta Khare, Kaiwen Zhang, Hongyang Sun, Aniruddha Gokhale, Julien Gascon-Samson, Yogesh Barve, Anirban Bhattacharjee and Xenofon Koutsoukos. 'Linearize, predict and place: Minimizing the Makespan for edge-based stream processing of directed acyclic graphs'. In: *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing, SEC 2019* (2019), pp. 1–14. DOI: 10.1145/3318216.3363315.

[88] Shanhe Yi, Zijiang Hao, Qingyang Zhang, Quan Zhang, Weisong Shi and Qun Li. 'Lavea'. In: *Proceedings of the Second ACM/IEEE Symposium on Edge Computing - SEC '17* (2017), pp. 1–13. DOI: 10.1145/3132211.3134459.

[89] Joel Wolf, Nikhil Bansal, Kirsten Hildrum, Sujay Parekh and Deepak Rajan. 'SODA : An Optimizing Scheduler for Large-Scale Stream-Based Distributed Computer Systems'. In: *Middleware 2008* (2008), pp. 306–325.

[90] Bin Cheng, Gürkan Solmaz, Flavio Cirillo, Ernö Kovacs, Kazuyuki Terasawa and Atsushi Kitazawa. 'FogFlow: Easy Programming of IoT Services Over Cloud and Edges for Smart Cities'. In: *IEEE Internet of Things Journal* 5.2 (2018), pp. 696–707. DOI: 10.1109/JIOT.2017.2747214.

[91] Tariq Qayyum, Asad Waqar Malik, Muazzam A Khan Khattak and Senior Member. 'FogNetSim ++ : A Toolkit for Modeling and Simulation of Distributed Fog Environment'. In: *IEEE Access* 6 (2018), pp. 63570–63583. DOI: 10.1109/ACCESS.2018.2877696.

[92] Harshit Gupta, Amir Vahid Dastjerdi, Soumya K. Ghosh and Rajkumar Buyya. 'iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments'. In: *Software - Practice and Experience* 47.9 (2017), pp. 1275–1296. DOI: 10.1002/spe.2509. arXiv: 1606.02007.

[93] Ye Xia, Xavier Etchevers, Loïc Letondeur, Thierry Coupaye and Frédéric Desprez. 'Combining hardware nodes and software components ordering-based heuristics

for optimizing the placement of distributed IoT applications in the fog'. In: *Proceedings of the ACM Symposium on Applied Computing* (2018), pp. 751–760. DOI: 10.1145/3167132.3167215.

[94] Yang Zhang, Dusit Niyato and Ping Wang. 'Offloading in Mobile Cloudlet Systems with Intermittent Connectivity'. In: *IEEE Transactions on Mobile Computing* 14.12 (2015), pp. 2516–2529. DOI: 10.1109/TMC.2015.2405539.

[95] Xinchen Lyu, Hui Tian, Li Jiang, Alexey Vinel, Sabita Maharjan, Stein Gjessing and Yan Zhang. 'Selective Offloading in Mobile Edge Computing for the Green Internet of Things'. In: *IEEE Network* 32.1 (2018), pp. 54–60. DOI: 10.1109/MNET.2018.1700101. arXiv: 1703.10750.

[96] Pegah Gazori, Dadmehr Rahbari and Mohsen Nickray. 'Saving time and cost on the scheduling of fog-based IoT applications using deep reinforcement learning approach'. In: *Future Generation Computer Systems* (2019). DOI: 10.1016/j.future.2019.09.060.

[97] Eman AbdElhalim, Marwa Obayya and Sherif Kishk. 'Distributed Fog-to-Cloud computing system: A minority game approach'. In: *Concurrency Computation* 31.15 (2019), pp. 1–18. DOI: 10.1002/cpe.5162.

[98] Juan Liu, Yuyi Mao, Jun Zhang and Khaled B Letaief. 'Delay-optimal computation task scheduling for mobile-edge computing systems'. In: *IEEE International Symposium on Information Theory*. Vol. 2016-Augus. 2016, pp. 1451–1455. DOI: 10.1109/ISIT.2016.7541539. arXiv: arXiv:1604.07525v3.

[99] Apostolos Galanopoulos, George Iosifidis and Theodoros Salonidis. 'Optimizing data analytics in energy constrained IoT networks'. In: *2018 16th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks, WiOpt 2018* (2018). DOI: 10.23919/WIOPT.2018.8362855.

[100] Dawei Sun, Guangyan Zhang, Songlin Yang, Weimin Zheng, Samee U. Khan and Keqin Li. 'Re-Stream: Real-time and energy-efficient resource scheduling in big data stream computing environments'. In: *Information Sciences* 319 (2015), pp. 92–112. DOI: 10.1016/j.ins.2015.03.027.

[101] Hong Liang Li, Jie Wu, Zhen Jiang, Xiang Li and Xiao Hui Wei. 'A Task Allocation Method for Stream Processing with Recovery Latency Constraint'. In: *Journal of Computer Science and Technology* 33.6 (2018), pp. 1125–1139. DOI: 10.1007/s11390-018-1876-6.

[102] Björn Lohrmann, Daniel Warneke and Odej Kao. 'Nephele streaming: Stream processing under QoS constraints at scale'. In: *Cluster Computing* 17.1 (2014), pp. 61–78. DOI: 10.1007/s10586-013-0281-8. arXiv: arXiv:1308.1031v1.

[103] Ruilong Deng, Rongxing Lu, Chengzhe Lai, Tom H Luan and Hao Liang. 'Optimal Workload Allocation in Fog-Cloud Computing Toward Balanced Delay and Power Consumption'. In: *IEEE Internet of Things Journal*. Vol. 3. 6. 2016, pp. 1171–1181. DOI: 10.1109/JIOT.2016.2565516.

[104] Sai Saketh Nandan Perala, Ioannis Galanis and Iraklis Anagnostopoulos. 'Fog Computing and Efficient Resource Management in the era of Internet-of-Video Things (IoVT)'. In: *Proceedings - IEEE International Symposium on Circuits and Systems* 2018-May (2018). DOI: 10.1109/ISCAS.2018.8351341.

[105] Anil Singh, Nitin Auluck, Omer Rana, Andrew Jones and Surya Nepal. 'RT-SANE : Real time security aware scheduling on the network edge'. In: *UCC 2017 - Proceedings of the10th International Conference on Utility and Cloud Computing* (2017), pp. 131–140. DOI: 10.1145/3147213.3147216.

[106] Salvatore Venticinque and Alba Amato. 'A methodology for deployment of IoT application in fog'. In: *Journal of Ambient Intelligence and Humanized Computing* 10.5 (2019), pp. 1955–1976. DOI: 10.1007/s12652-018-0785-4.

[107] Cory Thoma, Alexandros Labrinidis and Adam J. Lee. 'Automated operator placement in distributed Data Stream Management Systems subject to user constraints'. In: *Proceedings - International Conference on Data Engineering*. 2014, pp. 310–316. DOI: 10.1109/ICDEW.2014.6818346.

[108] Joshua E Siegel, Sumeet Kumar and Sanjay E Sarma. 'The Future Internet of Things : Secure ,' in: 5.4 (2018), pp. 2386–2398.

[109] Pierre Emmanuel Hladik, Hadrien Cambazard, Anne Marie Déplanche and Narendra Jussien. 'Solving a real-time allocation problem with constraint programming'. In: *Journal of Systems and Software* 81.1 (2008), pp. 132–149. DOI: 10.1016/j.jss.2007.02.032.

[110] Pushkara Ravindra, Aakash Khochare, Siva Prakash Reddy, Sarthak Sharma, Prateeksha Varshney and Yogesh Simmhan. 'ECHO: An adaptive orchestration platform for hybrid dataflows across cloud and edge'. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 10601 LNCS (2017), pp. 395–410. DOI: 10.1007/978-3-319-69035-3_28. arXiv: 1707.00889.

[111] Amir M. Rahmani, Tuan Nguyen Gia, Behailu Negash, Arman Anzanpour, Iman Azimi, Mingzhe Jiang and Pasi Liljeberg. 'Exploiting smart e-Health gateways at the edge of healthcare Internet-of-Things: A fog computing approach'. In: *Future Generation Computer Systems* 78 (2018), pp. 641–658. DOI: 10.1016/j.future.2017.02.014.

[112] Emma Fitzgerald, Michał Pióro and Artur Tomaszewski. 'Energy-Optimal Data Aggregation and Dissemination for the Internet of Things'. In: *IEEE Internet of Things Journal* 5.2 (2018), pp. 955–969. DOI: 10.1109/JIOT.2018.2803792.

[113] Jing Yue, Ming Xiao and Zhibo Pang. 'Distributed Fog Computing Based on Batched Sparse Codes for Industrial Control'. In: *IEEE Transactions on Industrial Informatics* 14.10 (2018), pp. 4683–4691. DOI: 10.1109/TII.2018.2857203.

[114] Ryuji Oma, Shigenari Nakamura, Dilawaer Duolikun, Tomoya Enokido and Makoto Takizawa. *Fault-tolerant fog computing models in the IOT*. Vol. 24. Springer International Publishing, 2019, pp. 14–25. DOI: 10.1007/978-3-030-02607-3_2.

[115] Saeid Dehnavi, Hamid Reza Faragardi, Mehdi Kargahi and Thomas Fahringer. 'A reliability-aware resource provisioning scheme for real-time industrial applications in a Fog-integrated smart factory'. In: *Microprocessors and Microsystems* 70 (2019), pp. 1–14. DOI: 10.1016/j.micpro.2019.05.011.

[116] Punit Rathore, Aravinda S. Rao, Sutharshan Rajasegarar, Elena Vanz, Jayavardhana Gubbi and Marimuthu Palaniswami. 'Real-Time Urban Microclimate Analysis Using Internet of Things'. In: *IEEE Internet of Things Journal* 5.2 (2018), pp. 500–511. DOI: 10.1109/JIOT.2017.2731875.

[117] Oresti Banos, Rafael Garcia, Juan A Holgado-terriza, Miguel Damas, Hector Pomares, Ignacio Rojas, Alejandro Saez and Claudia Villalonga. 'mHealthDroid : A Novel Framework for Agile Development of Mobile Health Applications'. In: *International Workshop on Ambient Assisted Living*. 2014, pp. 91–98. DOI: 10.1007/978-3-319-13105-4_14.

[118] Farshad Firouzi, Amir M. Rahmani, K. Mankodiya, M. Badaroglu, G. V. Merrett, P. Wong and Bahar Farahani. 'Internet-of-Things and big data for smarter healthcare: From device to architecture, applications and analytics'. In: *Future Generation Computer Systems* 78 (2018), pp. 583–586. DOI: 10.1016/j.future.2017.09.016.

[119] Antonio Brogi, Stefano Forti and Ahmad Ibrahim. 'How to Best Deploy Your Fog Applications, Probably'. In: *Proceedings - 2017 IEEE 1st International Conference on Fog and Edge Computing, ICFEC 2017* (2017), pp. 105–114. DOI: 10.1109/ICFEC.2017.8.

[120] Federico Concone, Giuseppe Lo Re and Marco Morana. 'A fog-based application for human activity recognition using personal smart devices'. In: *ACM Transactions on Internet Technology* 19.2 (2019). DOI: 10.1145/3266142.

[121] Li Su and Yongluan Zhou. 'Passive and Partially Active Fault Tolerance for Massively Parallel Stream Processing Engines'. In: *IEEE Transactions on Knowledge and Data Engineering* 31.1 (2019), pp. 32–45. DOI: 10.1109/TKDE.2017.2720602.

[122] Tiziano De Matteis and Gabriele Mencagli. 'Proactive elasticity and energy awareness in data stream processing'. In: *Journal of Systems and Software* 127 (2017), pp. 302–319. DOI: 10.1016/j.jss.2016.08.037.

[123] Jielong Xu, Zhenhua Chen, Jian Tang and Sen Su. 'T-storm: Traffic-aware online scheduling in storm'. In: *Proceedings - International Conference on Distributed Computing Systems* (2014), pp. 535–544. DOI: 10.1109/ICDCS.2014.61.

[124] Firat Karatas and Ibrahim Korpeoglu. 'Fog-Based Data Distribution Service (F-DAD) for Internet of Things (IoT) applications'. In: *Future Generation Computer Systems* 93 (2019), pp. 156–169. DOI: 10.1016/j.future.2018.10.039.

[125] Xavi Masip-Bruin, Eva Marin-Tordera, Admela Jukan and Guang Jie Ren. 'Managing resources continuity from the edge to the cloud: Architecture and performance'. In: *Future Generation Computer Systems* 79 (2018), pp. 777–785. DOI: 10.1016/j.future.2017.09.036.

[126] Antonio Brogi and Stefano Forti. 'QoS-aware deployment of IoT applications through the fog'. In: *IEEE Internet of Things Journal* 4.5 (2017), pp. 1–8. DOI: 10.1109/JIOT.2017.2701408.

[127] Engin Zeydan, Ejder Bastug, Mehdi Bennis, Manhal Abdel Kader, Ilyas Alper Karatepe, Ahmet Salih Er and Merouane Debbah. 'Big data caching for networking: Moving from cloud to edge'. In: *IEEE Communications Magazine* 54.9 (2016), pp. 36–42. DOI: 10.1109/MCOM.2016.7565185. arXiv: 1606.01581.

[128] Aras Can Onal, Omer Berat Sezer, Murat Ozbayoglu and Erdogan Dogdu. 'Weather data analysis and sensor fault detection using an extended IoT framework with semantics, big data, and machine learning'. In: *2017 IEEE International Conference on Big Data (Big Data)* (2017), pp. 2037–2046. DOI: 10.1109/BigData.2017.8258150.

[129] Michael Armbrust et al. *Above the Clouds: A Berkeley View of Cloud Computing*. Tech. rep. Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkeley, USA., 2009.

[130] Valeria Cardellini, Matteo Nardelli and Dario Luzi. 'Elastic stateful stream processing in storm'. In: *2016 International Conference on High Performance Computing and Simulation, HPCS 2016*. 2016, pp. 583–590. DOI: 10.1109/HPCSim.2016.7568388.

[131] Guoqiang Jerry Chen, Janet L Wiener, Shridhar Iyer, Anshul Jaiswal, Ran Lei, Nikhil Simha, Wei Wang, Kevin Wilfong, Tim Williamson and Serhat Yilmaz. 'Realtime Data Processing at Facebook'. In: *Facebook* (2016), pp. 1087–1098. DOI: 10.1145/2882903.2904441.

[132] Yugo Nakamura, Hirohiko Suwa, Yutaka Arakawa, Hirozumi Yamaguchi and Keiichi Yasumoto. 'Design and Implementation of Middleware for IoT Devices toward Real-Time Flow Processing'. In: *Proceedings - 2016 IEEE 36th International Conference on Distributed Computing Systems Workshops, ICDCSW 2016* (2016), pp. 162–167. DOI: 10.1109/ICDCSW.2016.37.

[133] Partha Pratim Ray, Dinesh Dash and Debashis De. 'Edge computing for Internet of Things: A survey, e-healthcare case study and future direction'. In: *Journal of Network and Computer Applications* 140.May (2019), pp. 1–22. DOI: 10.1016/j.jnca.2019.05.005.

[134] Mansoor Nasir, Khan Muhammad, Jaime Lloret, Arun Kumar Sangaiah and Muhammad Sajjad. 'Fog computing enabled cost-effective distributed summarization of surveillance videos for smart cities'. In: *Journal of Parallel and Distributed Computing* 126 (2019), pp. 161–170. DOI: 10.1016/j.jpdc.2018.11.004.

[135] Qiaozhi Xu and Junxing Zhang. 'PiFogBed: A Fog Computing Testbed Based on Raspberry Pi'. In: *2019 IEEE 38th International Performance Computing and Communications Conference, IPCCC 2019* (2019). DOI: 10.1109/IPCCC47392.2019.8958741.

[136] Jianhua He, Jian Wei, Kai Chen, Zuoyin Tang, Yi Zhou and Yan Zhang. 'Multitier Fog Computing With Large-Scale IoT Data Analytics for Smart Cities'. In: *IEEE Internet of Things Journal* 5.2 (2018), pp. 677–686. DOI: 10.1109/JIOT.2017.2724845.

[137] Claus Pahl, Sven Helmer, Lorenzo Miori, Julian Sanin and Brian Lee. 'A container-based edge cloud PaaS architecture based on raspberry Pi clusters'. In: *Proceedings - 2016 4th International Conference on Future Internet of Things and Cloud Workshops, W-FiCloud 2016* (2016), pp. 117–124. DOI: 10.1109/W-FiCloud.2016.36.

[138] Seyed Hossein Mortazavi, Mohammad Salehe, Carolina Simoes Gomes, Caleb Phillips and Eyal De Lara. 'CloudPath: A multi-Tier cloud computing framework'. In: *2017 2nd ACM/IEEE Symposium on Edge Computing, SEC 2017* (2017). DOI: 10.1145/3132211.3134464.

[139] *Apache Spark Streaming*. https://spark.apache.org/streaming/. (Accessed on 08/06/2020).

[140] Hwejoo Lee, Junghyun Oh, Kyungrae Kim and Hunje Yeon. 'A data streaming performance evaluation using resource constrained edge device'. In: *International Conference on Information and Communication Technology Convergence: ICT Convergence Technologies Leading the Fourth Industrial Revolution, ICTC 2017* 2017-December (2017), pp. 628–633. DOI: 10.1109/ICTC.2017.8191055.

[141] *Apache NiFi*. https://nifi.apache.org/. (Accessed on 08/06/2020).

[142] *Google Cloud Dataflow*. https://cloud.google.com/dataflow. (Accessed on 08/06/2020).

[143] *Internet of Things-IoT Platform- Australia — IBM Watson IoT*. https://www.ibm.com/au-en/internet-of-things. (Accessed on 08/06/2020).

[144] Jiayue Liang, Fang Liu, Shen Li and Zhenhua Cai. *A Comparative Research on Open Source Edge Computing Systems*. Vol. 11633 LNCS. Springer International Publishing, 2019, pp. 157–170. DOI: 10.1007/978-3-030-24265-7_14.

[145] Michael R Garey and David S Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. Vol. 174. W. H. Freeman and Company, 1979.

[146] Ruozhou Yu, Guoliang Xue and Xiang Zhang. 'Application Provisioning in FOG Computing-enabled Internet-of-Things: A Network Perspective'. In: *Proceedings - IEEE INFOCOM* 2018-April (2018), pp. 783–791. DOI: 10.1109/INFOCOM.2018.8486269.

[147] Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck and Guido Tack. 'MiniZinc: Towards a standard CP modelling language'. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 4741 LNCS (2007), pp. 529–543. DOI: 10.1007/978-3-540-74970-7_38.

[148] Peter J. Stuckey, Thibaut Feydy, Andreas Schutt, Guido Tack and Julien Fischer. 'The MiniZinc challenge 2008-2013'. In: *AI Magazine* 35.2 (2014), pp. 55–60. DOI: 10.1609/aimag.v35i2.2539.

[149] Charles Prud'homme, Jean-Guillaume Fages and Xavier Lorca. *Choco Solver Documentation*. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S. 2016.

[150] *CPLEX Optimizer — IBM*. https://www.ibm.com/analytics/cplex-optimizer. (Accessed on 07/28/2020).

[151] Karima Velasquez, David Perez Abreu, Marilia Curado and Edmundo Monteiro. 'Service placement for latency reduction in the internet of things'. In: *Annales des Telecommunications/Annals of Telecommunications* 72.1-2 (2017), pp. 105–115. DOI: 10.1007/s12243-016-0524-9.

[152] Hua Jun Hong, Pei Hsuan Tsai, An Chieh Cheng, Md Yusuf Sarwar Uddin, Nalini Venkatasubramanian and Cheng Hsin Hsu. 'Supporting Internet-of-Things Analytics in a Fog Computing Platform'. In: *Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom* 2017-December (2017), pp. 138–145. DOI: 10.1109/CloudCom.2017.45.

[153] Ashkan Yousefpour, Genya Ishigaki, Riti Gour and Jason P. Jue. 'On Reducing IoT Service Delay via Fog Offloading'. In: *IEEE Internet of Things Journal* 5.2 (2018), pp. 998–1010. DOI: 10.1109/JIOT.2017.2788802. arXiv: 1804.07376.

[154] Onur Ascigil, Truong Khoa Phan, Argyrios G. Tasiopoulos, Vasilis Sourlas, Ioannis Psaras and George Pavlou. 'On Uncoordinated Service Placement in Edge-Clouds'. In: *Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom* 2017-December (2017), pp. 41–48. DOI: 10.1109/CloudCom.2017.46.

[155] Lin Gu, Deze Zeng, Song Guo, Ahmed Barnawi and Yong Xiang. 'Cost efficient resource management in fog computing supported medical cyber-physical system'. In: *IEEE Transactions on Emerging Topics in Computing* 5.1 (2017), pp. 108–119. DOI: 10.1109/TETC.2015.2508382.

[156] Bowen Zhou, Amir Vahid Dastjerdi, Rodrigo N Calheiros and Rajkumar Buyya. 'An online algorithm for task offloading in heterogeneous mobile clouds'. In: *ACM Transactions on Internet Technology* 18.2 (2018). DOI: 10.1145/3122981.

[157] Jiao Zhang, Xiping Hu, Zhaolong Ning, Edith C.H. Ngai, Li Zhou, Jibo Wei, Jun Cheng and Bin Hu. 'Energy-latency tradeoff for energy-aware offloading in mobile edge computing networks'. In: *IEEE Internet of Things Journal* 5.4 (2018), pp. 2633–2645. DOI: 10.1109/JIOT.2017.2786343.

[158] Michael J. Neely. *Stochastic network optimization with application to communication and queueing systems*. Vol. 7. 2010, pp. 1–199.

[159] Lu Hou, Kan Zheng, Periklis Chatzimisios and Yi Feng. 'A Continuous-Time Markov decision process-based resource allocation scheme in vehicular cloud for mobile video services'. In: *Computer Communications* 118.September 2017 (2018), pp. 140–147. DOI: 10.1016/j.comcom.2017.10.011.

[160] Shiqiang Wang, Rahul Urgaonkar, Murtaza Zafer, Ting He, Kevin Chan and Kin K. Leung. 'Dynamic Service Migration in Mobile Edge Computing Based on Markov Decision Process'. In: *IEEE/ACM Transactions on Networking* 27.3 (2019), pp. 1272–1288. DOI: 10.1109/TNET.2019.2916577. arXiv: arXiv:1506.05261v2.

[161] Mati B. Terefe, Heezin Lee, Nojung Heo, Geoffrey C. Fox and Sangyoon Oh. 'Energy-efficient multisite offloading policy using Markov decision process for mobile cloud computing'. In: *Pervasive and Mobile Computing* 27 (2016), pp. 75–89. DOI: 10.1016/j.pmcj.2015.10.008.

[162] Dongqing Liu, Lyes Khoukhi and Abdelhakim Hafid. 'Data offloading in mobile cloud computing: A Markov Decision Process approach'. In: *IEEE International Conference on Communications* (2017), pp. 1–6. DOI: 10.1109/ICC.2017.7997070.

[163] Adlen Ksentini, Tarik Taleb and Min Chen. 'A Markov Decision Process-based service migration procedure for follow me cloud'. In: *2014 IEEE International Conference on Communications, ICC 2014* (2014), pp. 1350–1354. DOI: 10.1109/ICC.2014.6883509.

[164] Shiqiang Wang, Rahul Urgaonkar, Ting He, Murtaza Zafer, Kevin Chan and Kin K. Leung. 'Mobility-induced service migration in mobile micro-clouds'. In: *Proceedings - IEEE Military Communications Conference MILCOM* (2014), pp. 835–840. DOI: 10.1109/MILCOM.2014.145.

[165] G. Oddi, M. Panfili, A. Pietrabissa, L. Zuccaro and V. Suraci. 'A resource allocation algorithm of multi-cloud resources based on Markov Decision Process'. In: *Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom* 1 (2013), pp. 130–135. DOI: 10.1109/CloudCom.2013.24.

[166] Juan Liu, Yuyi Mao, Jun Zhang and Khaled B Letaief. 'Delay-optimal computation task scheduling for mobile-edge computing systems'. In: *IEEE International Symposium on Information Theory*. Vol. 2016-Augus. 2016, pp. 1451–1455. DOI: 10.1109/ISIT.2016.7541539. arXiv: arXiv:1604.07525v3.

[167] Lei Lei, Huijuan Xu, Xiong Xiong, Kan Zheng and Wei Xiang. 'Joint computation offloading and multiuser scheduling using approximate dynamic programming in NB-IoT edge computing system'. In: *IEEE Internet of Things Journal* 6.3 (2019), pp. 5345–5362. DOI: 10.1109/JIOT.2019.2900550.

[168] Gabriele Russo Russo, Valeria Cardellini and Francesco Lo Presti. 'Reinforcement Learning Based Policies for Elastic Stream Processing on Heterogeneous Resources'. In: *Proceedings of the 13th ACM International Conference on Distributed and Event-based Systems* (2019), pp. 31–42. DOI: 10.1145/3328905.3329506.

[169] Gabriele Russo Russo, Matteo Nardelli, Valeria Cardellini and Francesco Lo Presti. 'Multi-Level Elasticity for Wide-Area Data Streaming Systems: A Reinforcement Learning Approach'. In: *Algorithms* 11.9 (2018), p. 134. DOI: 10.3390/a11090134.

[170] Md Golam Rabiul Alam, Mohammad Mehedi Hassan, Md ZIa Uddin, Ahmad Almogren and Giancarlo Fortino. 'Autonomic computation offloading in mobile edge for IoT applications'. In: *Future Generation Computer Systems* 90 (2019), pp. 149–157. DOI: 10.1016/j.future.2018.07.050.

[171] Xianfu Chen, Honggang Zhang, Celimuge Wu, Shiwen Mao, Yusheng Ji and Medhi Bennis. 'Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning'. In: *IEEE Internet of Things Journal* 6.3 (2019), pp. 4005–4018. DOI: 10.1109/JIOT.2018.2876279.

[172] Xu Chen, Lei Jiao, Wenzhong Li and Xiaoming Fu. 'Efficient Multi-User Computation Offloading for Mobile-Edge Cloud Computing'. In: *IEEE/ACM Transactions on Networking* 24.5 (2016), pp. 2795–2808. DOI: 10.1109/TNET.2015.2487344. arXiv: 1510.00888.

[173] Lina Ni, Jinquan Zhang and Jiguo Yu. 'Priced timed petri nets based resource allocation strategy for fog computing'. In: *Proceedings - 2016 International Conference on Identification, Information and Knowledge in the Internet of Things, IIKI 2016* 2018-January.5 (2018), pp. 39–44. DOI: 10.1109/IIKI.2016.87.

[174] Liqing Liu, Zheng Chang, Senior Member, Xijuan Guo, Shiwen Mao and Senior Member. 'Multiobjective Optimization for Computation Offloading in Fog Computing'. In: *IEEE Internet of Things Journal* 5.1 (2018), pp. 283–294. DOI: 10.1109/JIOT.2017.2780236.

[175] Kezhi Wang, Kun Yang, Xinhou Wang and Chathura Sarathchandra Magurawalage. 'Cost-effective resource allocation in C-RAN with mobile cloud'. In: *2016 IEEE International Conference on Communications, ICC 2016* (2016). DOI: 10.1109/ICC.2016.7511044.

[176] Hamed Shah-Mansouri and Vincent W.S. Wong. 'Hierarchical Fog-Cloud Computing for IoT Systems: A Computation Offloading Game'. In: *IEEE Internet of Things Journal* 5.4 (2018), pp. 3246–3257. DOI: 10.1109/JIOT.2018.2838022. arXiv: 1710.06089.

[177] Min Chen, Wei Li, Giancarlo Fortino, Yixue Hao, Long Hu and Iztok Humar. 'A dynamic service migration mechanism in edge cognitive computing'. In: *ACM Transactions on Internet Technology* 19.2 (2019). DOI: 10.1145/3239565.

[178] Zhaolong Ning, Peiran Dong, Xiaojie Wang, Joel J.P.C. Rodrigues and Feng Xia. 'Deep reinforcement learning for vehicular edge computing: An intelligent offloading system'. In: *ACM Transactions on Intelligent Systems and Technology* 10.6 (2019). DOI: 10.1145/3317572.

[179] Gilsoo Lee, Walid Saad and Mehdi Bennis. 'An online optimization framework for distributed fog network formation with minimal latency'. In: *IEEE Transactions on Wireless Communications* 18.4 (2019), pp. 2244–2258. DOI: 10.1109/TWC.2019.2901850. arXiv: 1710.05239.

[180] Thomas Heinze, Zbigniew Jerzak, Gregor Hackenbroich and Christof Fetzer. 'Latency-aware elastic scaling for distributed data stream processing systems'. In: *DEBS 2014 - Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems* (2014), pp. 13–22. DOI: 10.1145/2611286.2611294.

[181] Ashkan Yousefpour, Ashish Patil, Genya Ishigaki, Inwoong Kim, Xi Wang, Hakki C. Cankaya, Qiong Zhang, Weisheng Xie and Jason P. Jue. 'FOGPLAN: A lightweight QoS-aware dynamic fog service provisioning framework'. In: *IEEE Internet of Things Journal* 6.3 (2019), pp. 5080–5096. DOI: 10.1109/JIOT.2019.2896311.

[182] Yan Sun, Fuhong Lin and Haitao Xu. *Multi-objective Optimization of Resource Scheduling in Fog Computing Using an Improved NSGA-II*. 2018. DOI: 10.1007/s11277-017-5200-5.

[183] Shuang Chen, Yanzhi Wang and Massoud Pedram. 'A semi-Markovian decision process based control method for offloading tasks from mobile devices to the cloud'. In: *GLOBECOM - IEEE Global Telecommunications Conference* (2013), pp. 2885–2890. DOI: 10.1109/GLOCOM.2013.6831512.

[184] Redowan Mahmud, Kotagiri Ramamohanarao and Rajkumar Buyya. 'Latency-aware application module management for fog computing environments'. In: *ACM Transactions on Internet Technology* 19.1 (2018). DOI: 10.1145/3186592.

[185] Huaqing Zhang, Yong Xiao, Shengrong Bu, Dusit Niyato, F. Richard Yu and Zhu Han. 'Computing Resource Allocation in Three-Tier IoT Fog Networks: A Joint Optimization Approach Combining Stackelberg Game and Matching'. In: *IEEE Internet of Things Journal* 4.5 (2017), pp. 1204–1215. DOI: 10.1109/JIOT.2017.2688925. arXiv: 1701.03922.

[186] Tao Ouyang, Zhi Zhou and Xu Chen. 'Follow Me at the Edge: Mobility-Aware Dynamic Service Placement for Mobile Edge Computing'. In: *IEEE Journal on Selected Areas in Communications* 36.10 (2018), pp. 2333–2345. DOI: 10.1109/JSAC.2018.2869954. arXiv: arXiv:1809.05239v1.

[187] William Tärneberg, Amardeep Mehta, Eddie Wadbro, Johan Tordsson, Johan Eker, Maria Kihl and Erik Elmroth. 'Dynamic application placement in the Mobile Cloud Network'. In: *Future Generation Computer Systems* 70 (2017), pp. 163–177. DOI: 10.1016/j.future.2016.06.021.

[188] Ram Govinda Aryal and Jorn Altmann. 'Dynamic application deployment in federations of clouds and edge resources using a multiobjective optimization AI algorithm'. In: *2018 3rd International Conference on Fog and Mobile Edge Computing, FMEC 2018* (2018), pp. 147–154. DOI: 10.1109/FMEC.2018.8364057.

[189] Vincenzo Scoca, Atakan Aral, Ivona Brandic, Rocco De Nicola and Rafael Brundo Uriarte. 'Scheduling latency-sensitive applications in edge computing'. In: *CLOSER 2018 - Proceedings of the 8th International Conference on Cloud Computing and Services Science* 2018-January.Closer 2018 (2018), pp. 158–168. DOI: 10.5220/0006706201580168.

[190] Manisha Luthra, Boris Koldehofe, Pascal Weisenburger, Guido Salvaneschi and Raheel Arif. 'TCEP: Adapting to dynamic user environments by enabling transitions between operator placement mechanisms'. In: *Proceedings of the 12th ACM International Conference on Distributed and Event-based Systems*. 2018, pp. 136–147.

[191] Mohammad M. Shurman and Maha K. Aljarah. 'Collaborative execution of distributed mobile and IoT applications running at the edge'. In: *2017 International Conference on Electrical and Computing Technologies and Applications, ICECTA 2017* 2018-Janua (2017), pp. 1–5. DOI: 10.1109/ICECTA.2017.8252057.

[192] Nitinder Mohan, Pengyuan Zhou, Keerthana Govindaraj and Jussi Kangasharju. 'Managing data in computational edge clouds'. In: *MECOMM 2017 - Proceedings of the 2017 Workshop on Mobile Edge Communications, Part of SIGCOMM 2017* (2017), pp. 19–24. DOI: 10.1145/3098208.3098212.

[193] Xuan Qui Pham and Eui Nam Huh. 'Towards task scheduling in a cloud-fog computing system'. In: *18th Asia-Pacific Network Operations and Management Symposium, APNOMS 2016: Management of Softwarized Infrastructure - Proceedings* (2016). DOI: 10.1109/APNOMS.2016.7737240.

[194] Atakan Aral and Tolga Ovatman. 'A decentralized replica placement algorithm for edge computing'. In: *IEEE Transactions on Network and Service Management* 15.2 (2018), pp. 516–529. DOI: 10.1109/TNSM.2017.2788945.

[195] Mennan Selimi, Davide Vega, Felix Freitag and Luís Veiga. 'Towards network-aware service placement in community network micro-clouds'. In: *European Conference on Parallel Processing*. Springer. 2016, pp. 376–388.

[196] Fangxiaoqi Yu, Haopeng Chen and Jinqing Xu. 'DMPO : Dynamic mobility-aware partial offloading in mobile edge computing'. In: *Future Generation Computer Systems* 89 (2018), pp. 722–735. DOI: 10.1016/j.future.2018.07.032.

[197] Kasper Grud Skat Madsen and Yongluan Zhou. 'Dynamic resource management in a Massively Parallel Stream Processing Engine'. In: *International Conference on Information and Knowledge Management, Proceedings*. Vol. 19-23-Oct-. 2015, pp. 13–22. DOI: 10.1145/2806416.2806449.

[198] Ruijuan Zheng, Kang Liu, Junlong Zhu, Mingchuan Zhang and Qingtao Wu. 'Stochastic resource scheduling via bilayer dynamic Markov decision process in mobile cloud networks'. In: *Computer Communications* 145.July (2019), pp. 234–242. DOI: 10.1016/j.comcom.2019.07.004.

[199] Mohammad Aazam and Eui Nam Huh. 'Fog computing micro datacenter based dynamic resource estimation and pricing model for IoT'. In: *Proceedings - International Conference on Advanced Information Networking and Applications, AINA* 2015-April.March (2015), pp. 687–694. DOI: 10.1109/AINA.2015.254.

[200] Daniele Foroni, Cristian Axenie, Stefano Bortoli, Mohamad Al Hajj Hassan, Ralph Acker, Radu Tudoran, Goetz Brasche and Yannis Velegrakis. 'Moira: A goal-oriented incremental machine learning approach to dynamic resource cost estimation in distributed stream processing systems'. In: *ACM International Conference Proceeding Series* (2018). DOI: 10.1145/3242153.3242160.

[201] Jianyu Wang, Jianli Pan and Flavio Esposito. 'Elastic urban video surveillance system using edge computing'. In: *SmartIoT 2017 - Proceedings of the Workshop on Smart Internet of Things* (2017). DOI: 10.1145/3132479.3132490.

[202] Huayong Wang, Li Shiuan Peh, Emmanouil Koukoumidis, Shao Tao and Mun Choon Chan. 'Meteor shower: A reliable stream processing system for commodity data centers'. In: *Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium, IPDPS 2012* (2012), pp. 1180–1191. DOI: 10.1109/IPDPS.2012.108.

[203] He Li, Kaoru Ota and Mianxiong Dong. 'Deep reinforcement scheduling for mobile crowdsensing in fog computing'. In: *ACM Transactions on Internet Technology* 19.2 (2019). DOI: 10.1145/3234463.

[204] Sean T. Allen, Matthew Jankowski and Peter Pathirana. *Storm Applied: Strategies for Real-Time Event Processing*. 1st. USA: Manning Publications Co., 2015.

[205] Xunyun Liu and Rajkumar Buyya. 'D-Storm: Dynamic resource-efficient scheduling of stream processing applications'. In: *Proceedings of the International Conference on Parallel and Distributed Systems - ICPADS* 2017-Decem (2018), pp. 485–492. DOI: 10.1109/ICPADS.2017.00070.

[206] Albert Jonathan, Abhishek Chandra and Jon Weissman. 'Rethinking adaptability in wide-area stream processing systems'. In: *10th USENIX Workshop on Hot Topics in Cloud Computing, HotCloud 2018, co-located with USENIX ATC 2018*. 2018.

[207] Olena Skarlat, Stefan Schulte, Michael Borkowski and Philipp Leitner. 'Resource provisioning for IoT services in the fog'. In: *Proceedings - 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications, SOCA 2016* (2016), pp. 32–39. DOI: 10.1109/SOCA.2016.10.

[208] Abhishek Tiwari, Brian Ramprasad, Seyed Hossein Mortazavi, Moshe Gabel and Eyal De Lara. 'Reconfigurable streaming for the mobile edge'. In: *HotMobile 2019 - Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications* (2019), pp. 153–158. DOI: 10.1145/3301293.3302355.

[209] Daniel Balouek et al. 'Adding Virtualization Capabilities to the Grid'5000 Testbed'. In: *Cloud Computing and Services Science*. Ed. by Ivan I. Ivanov, Marten van Sinderen, Frank Leymann and Tony Shan. Vol. 367. Communications in Computer and Information Science. Springer International Publishing, 2013, pp. 3–20. DOI: 10.1007/978-3-319-04519-1_1.

[210] Richard Olaniyan, Olamilekan Fadahunsi, Muthucumaru Maheswaran and Mohamed Faten Zhani. 'Opportunistic Edge Computing: Concepts, Opportunities and Research Challenges'. In: i (2018). arXiv: arXiv:1806.04311v3.

[211] Subarna Chatterjee and Christine Morin. 'Experimental Study on the Performance and Resource Utilization of Data Streaming Frameworks'. In: *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)* (2018), pp. 143–152. DOI: 10.1109/CCGRID.2018.00029.

[212] Vania Marangozova-Martin, Noel De Palma and Ahmed El Rheddane. 'Multi-Level Elasticity for Data Stream Processing'. In: *IEEE Transactions on Parallel and Distributed Systems* 30.10 (2019), pp. 2326–2337. DOI: 10.1109/TPDS.2019.2907950.

[213] Cedric Adjih et al. 'FIT IoT-LAB: A large scale open experimental IoT testbed'. In: *IEEE World Forum on Internet of Things, WF-IoT 2015 - Proceedings* (2015), pp. 459–464. DOI: 10.1109/WF-IoT.2015.7389098.

[214] Georgios Z. Papadopoulos, Antoine Gallais, Guillaume Schreiner, Emery Jou and Thomas Noel. 'Thorough IoT testbed characterization: From proof-of-concept to repeatable experimentations'. In: *Computer Networks* 119 (2017), pp. 86–101. DOI: 10.1016/j.comnet.2017.03.012.

[215] Hend Gedawy, Sannan Tariq, Abderrahmen Mtibaa and Khaled Harras. 'Cumulus: A distributed and flexible computing testbed for edge cloud computational offloading'. In: *2016 Cloudification of the Internet of Things, CIoT 2016* 3 (2017), pp. 1–6. DOI: 10.1109/CIOT.2016.7872920.

[216] Huayong Wang and Li Shiuan Peh. 'Mobistreams: A reliable distributed stream processing system for mobile devices'. In: *Proceedings of the International Parallel and Distributed Processing Symposium, IPDPS*. 2014, pp. 51–60. DOI: 10.1109/IPDPS.2014.17.

[217] S S Ogden and T Guo. 'MODI: Mobile Deep Inference Made Efficient by Edge Computing'. In: *Usenix Workshop on Hot Topics in Edge Computing* 1 (2018).

[218] Jakub Dolezal, Zdenek Becvar and Tomas Zeman. 'Performance evaluation of computation offloading from mobile device to the edge of mobile network'. In: *2016 IEEE Conference on Standards for Communications and Networking, CSCN 2016* (2016), pp. 1–7. DOI: 10.1109/CSCN.2016.7785153.

[219] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César A.F. De Rose and Rajkumar Buyya. 'CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms'. In: *Software - Practice and Experience* 41.1 (2011), pp. 23–50. DOI: 10.1002/spe.995. arXiv: 0307245 [hep-th].

[220] Mohammed Islam Naas, Laurent Lemarchand, Jalil Boukhobza and Philippe Raipin. 'A graph partitioning-based heuristic for runtime IoT data placement strategies in a fog infrastructure'. In: *Proceedings of the ACM Symposium on Applied Computing* (2018), pp. 767–774. DOI: 10.1145/3167132.3167217.

[221] Redowan Mahmud, Fernando Luiz Koch and Rajkumar Buyya. 'Cloud-Fog Interoperability in IoT-enabled Healthcare Solutions'. In: *Proceedings of the 19th International Conference on Distributed Computing and Networking - ICDCN '18* December 2017 (2018), pp. 1–10. DOI: 10.1145/3154273.3154347.

[222] Enzo Baccarelli, Paola G.Vinueza Naranjo, Michele Scarpiniti, Mohammad Shojafar and Jemal H. Abawajy. 'Fog of Everything: Energy-Efficient Networked Computing Architectures, Research Challenges, and a Case Study'. In: *IEEE Access* 5 (2017), pp. 9882–9910. DOI: 10.1109/ACCESS.2017.2702013.

[223] Gaolei Li, Jun Wu, Jianhua Li, Kuan Wang and Tianpeng Ye. 'Service Popularity-based Smart Resources Partitioning for Fog Computing-enabled Industrial Internet of Things'. In: *IEEE Transactions on Industrial Informatics* 14.10 (2018), pp. 4702–4711. DOI: 10.1109/TII.2018.2845844.

[224] Mukhtar M.E. Mahmoud, Joel J.P.C. Rodrigues, Kashif Saleem, Jalal Al-Muhtadi, Neeraj Kumar and Valery Korotaev. 'Towards energy-aware fog-enabled cloud of things for healthcare'. In: *Computers and Electrical Engineering* 67 (2018), pp. 58–69. DOI: 10.1016/j.compeleceng.2018.02.047.

[225] Cagatay Sonmez, Atay Ozgovde and Cem Ersoy. 'EdgeCloudSim: An environment for performance evaluation of Edge Computing systems'. In: *2017 2nd International Conference on Fog and Mobile Edge Computing, FMEC 2017* (2017), pp. 39–44. DOI: 10.1109/FMEC.2017.7946405.

[226] Cheikh Saliou Mbacke Babou, Doudou Fall, Shigeru Kashihara, Ibrahima Niang and Youki Kadobayashi. 'Home edge computing (HEC): design of a new edge computing technology for achieving ultra-low latency'. In: *International conference on edge computing*. Springer. 2018, pp. 3–17.

[227] András Varga and Rudolf Hornig. 'An overview of the OMNeT++ simulation environment'. In: *SIMUTools 2008 - 1st International ICST Conference on Simulation Tools and Techniques for Communications, Networks and Systems*. August. 2008. DOI: 10.4108/ICST.SIMUTOOLS2008.3027.

[228] Piotr Borylo, Artur Lason, Jacek Rzasa, Andrzej Szymanski and Andrzej Jajszczyk. 'Energy-aware fog and cloud interplay supported by wide area software defined networking'. In: *2016 IEEE International Conference on Communications, ICC 2016* (2016). DOI: 10.1109/ICC.2016.7511451.

[229] Tommaso Cucinotta, Aram Santogidis and Bell Laboratories. 'CloudNetSim - Simulation of Real-Time Cloud Computing Applications'. In: *Proceedings of the 4th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2013)* (2013), pp. 1–6.

[230] Asad Malik, Kashif Bilal, Saif Malik, Zahid Anwar, Khurram Aziz, Dzmitry Kliazovich, Nasir Ghani, Samee Khan and Rajkumar Buyya. 'CloudNetSim++: A GUI Based Framework for Modeling and Simulation of Data Centers in OMNeT++'. In: *IEEE Transactions on Services Computing* 1374.c (2015), pp. 1–1. DOI: 10.1109/TSC.2015.2496164.

[231] Teerawat Issariyakul and Ekram Hossain. 'Introduction to network simulator 2 (NS2)'. In: *Introduction to network simulator NS2*. Springer, 2009, pp. 1–18.

[232] Jefferson Morales, Erika Rosas and Nicolas Hidalgo. 'Symbiosis: Sharing mobile resources for stream processing'. In: *Proceedings - International Symposium on Computers and Communications*. Vol. Workshops. 2014. DOI: 10.1109/ISCC.2014.6912641.

[233] Arslan Munir, Joseph Antoon and Ann Gordon-Ross. 'Modeling and Analysis of Fault Detection and Fault Tolerance in Wireless Sensor Networks'. In: *ACM Transactions on Embedded Computing Systems* 14.1 (2015), pp. 1–43. DOI: 10.1145/2680538.

[234] Lorenzo Saino, Ioannis Psaras and George Pavlou. 'Icarus: A Caching simulator for Information Centric Networking (ICN)'. In: *SIMUTools 2014 - 7th International Conference on Simulation Tools and Techniques* (2014), pp. 66–75. DOI: 10.4108/icst.simutools.2014.254630.

[235] Ari Keränen, Jörg Ott and Teemu Kärkkäinen. 'The ONE simulator for DTN protocol evaluation'. In: *SIMUTools 2009 - 2nd International ICST Conference on Simulation Tools and Techniques* (2009). DOI: 10.4108/ICST.SIMUTOOLS2009.5674.

[236] Zdenek Becvar, Jan Plachy and Pavel Mach. 'Path selection using handover in mobile networks with cloud-enabled small cells'. In: *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC* 2014-June (2014), pp. 1480–1485. DOI: 10.1109/PIMRC.2014.7136402.

[237] Huang Xin, Sivakumar Ganapathy and Tilman Wolf. 'Evaluating algorithms for composable service placement in computer networks'. In: *IEEE International Conference on Communications* (2009). DOI: 10.1109/ICC.2009.5199007.

[238] Vrinda Kochar and Arnab Sarkar. 'Real time resource allocation on a dynamic two level symbiotic fog architecture'. In: *Proceedings - 2016 6th International Symposium on Embedded Computing and System Design, ISED 2016* (2017), pp. 49–55. DOI: 10.1109/ISED.2016.7977053.

[239] Isaac Lera, Carlos Guerrero and Carlos Juiz. 'Comparing centrality indices for network usage optimization of data placement policies in fog devices'. In: *2018 3rd International Conference on Fog and Mobile Edge Computing, FMEC 2018* (2018), pp. 115–122. DOI: 10.1109/FMEC.2018.8364053.

[240] Shiqiang Wang, Murtaza Zafer and Kin K. Leung. 'Online Placement of Multi-Component Applications in Edge Computing Environments'. In: *IEEE Access* 5 (2017), pp. 2514–2533. DOI: 10.1109/ACCESS.2017.2665971. arXiv: 1605.08023.

[241] Peter Pietzuch, Jonathan Ledlie, Jeffrey Shneidman, Mema Roussopoulos, Matt Welsh and Margo Seltzer. 'Network-aware operator placement for stream processing systems'. In: *Proceedings - International Conference on Data Engineering*. Vol. 2006. 2006, p. 49. DOI: 10.1109/ICDE.2006.105. arXiv: 9780201398298.

[242] Krittin Intharawijitr, Student Member, Katsuyoshi Iida and Senior Member. 'Simulation Study of Low Latency Network Architecture Using Mobile Edge Computing'. In: *IEICE TRANSACTIONS on Information and Systems* 5 (2017), pp. 963–972.

[243] Jun Huang, Guoquan Liu, Qiang Duan and Yuhong Yan. 'QoS-aware service composition for converged network-cloud service provisioning'. In: *Proceedings - 2014 IEEE International Conference on Services Computing, SCC 2014*. 2014, pp. 67–74. DOI: 10.1109/SCC.2014.18.

[244] Qiang Fan and Nirwan Ansari. 'Application Aware Workload Allocation for Edge Computing based IoT'. In: *IEEE Internet of Things Journal* 5.3 (2018), pp. 2146–2153. DOI: 10.1109/JIOT.2018.2826006.

[245] Chin Lung Hsu and Judy Chuan Chuan Lin. 'An empirical examination of consumer adoption of Internet of Things services: Network externalities and concern for information privacy perspectives'. In: *Computers in Human Behavior* 62 (2016), pp. 516–527. DOI: 10.1016/j.chb.2016.04.023. arXiv: arXiv:1011.1669v3.

[246] Daniele D Agostino, Lucia Morganti, Elena Corni, Daniele Cesini and Ivan Merelli. 'Combining Edge and Cloud Computing for Low-Power , Cost-Effective Metagenomics Analysis'. In: *Future Generation Computer Systems* (2018). DOI: 10.1016/j.future.2018.07.036.

[247] Rashmika Nawaratne, Damminda Alahakoon, Daswin De Silva, Prem Chhetri and Naveen Chilamkurti. 'Self-evolving intelligent algorithms for facilitating data interoperability in IoT environments'. In: *Future Generation Computer Systems* 86 (2018), pp. 421–432. DOI: 10.1016/j.future.2018.02.049.

[248] Biljana L Risteska Stojkoska and Kire V Trivodaliev. 'A review of Internet of Things for smart home : Challenges and solutions'. In: *Journal of Cleaner Production* 140 (2017), pp. 1454–1464. DOI: 10.1016/j.jclepro.2016.10.006.

[249] Gartner Inc., Mark Hung, Chet Geschickter, Nathan Nuttall, Jenny Beresford, Erik T. Heidt and Mike J. Walker. *Leading the IoT - Gartner Insights on How to Lead in a Connected World*. 2017, pp. 1–29.

[250] Md Golam Rabiul Alam, Mohammad Mehedi Hassan, Md. ZIa Uddin, Ahmad Almogren and Giancarlo Fortino. 'Autonomic computation offloading in mobile edge for IoT applications'. In: *Future Generation Computer Systems* 90 (2019), pp. 149–157. DOI: https://doi.org/10.1016/j.future.2018.07.050.

[251] Yun Chao Hu, Milan Patel, Dario Sabella, Nurit Sprecher and Valerie Young. *Mobile Edge Computing: A Key Technology Towards 5G*. Whitepaper ETSI White Paper No. 11. European Telecommunications Standards Institute (ETSI), 2015.

[252] Giovanni Nardini, Antonio Virdis, Giovanni Stea and Angelo Buono. 'SimuLTE-MEC: Extending SimuLTE for Multi-Access Edge Computing'. In: *Proceedings of the 5th International OMNeT++ Community Summit*. Ed. by Anna F\"orster, Asanga Udugama, Antonio Virdis and Giovanni Nardini. Vol. 56. EPiC Series in Computing. EasyChair, 2018, pp. 35–42. DOI: 10.29007/7g1p.

[253] Fabian Kaup. 'Energy-efficiency and Performance in Communication Networks'. In: (2017).

[254] Andrea Camesi, Jarle Hulaas and Walter Binder. 'Continuous bytecode instruction counting for CPU consumption estimation'. In: *Third International Conference on the Quantitative Evaluation of Systems, QEST 2006*. 2006, pp. 19–28. DOI: 10.1109/QEST.2006.12.

[255] Fabian Kaup, Philip Gottschling and David Hausheer. 'PowerPi: Measuring and modeling the power consumption of the Raspberry Pi'. In: *39th Annual IEEE Conference on Local Computer Networks*. 2014, pp. 236–243. DOI: 10.1109/LCN.2014.6925777.

[256] Antonio Virdis, Giovanni Stea and Giovanni Nardini. 'Simulating LTE/LTE-Advanced Networks with SimuLTE'. In: *Simulation and Modeling Methodologies, Technologies and Applications*. Ed. by Mohammad S Obaidat, Tuncer Ören, Janusz Kacprzyk and Joaquim Filipe. Cham: Springer International Publishing, 2015, pp. 83–105.

[257] Yogesh Simmhan, Saima Aman, Alok Kumbhare, Rongyang Liu, Sam Stevens, Qunzhi Zhou and Viktor Prasanna. 'Cloud-based software platform for big data analytics in smart grids'. In: *Computing in Science & Engineering* 15.4 (2013), pp. 38–47.

[258] Gecode Team. *Gecode: Generic Constraint Development Environment*. http://www.gecode.org. 2018.

[259] Cisco. 'Cisco Annual Internet Report (2018 - 2023)'. In: *Cisco* (2020), pp. 1–41.

[260] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. 1st. New York, NY, USA: John Wiley & Sons, Inc., 1994.

[261] Richard Bellman. 'A Markovian decision process'. In: *Journal of mathematics and mechanics* (1957), pp. 679–684.

[262] Tom Cooper, Paul Ezhilchelvan and Isi Mitrani. 'A Queuing Model of a Stream-Processing Server'. In: *2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE. 2019, pp. 27–35.

[263] Roberto Morabito, Vittorio Cozzolino, Aaron Yi Ding, Nicklas Beijar and Jörg Ott. 'Consolidate IoT Edge Computing with Lightweight Virtualization'. In: *IEEE Network* February (2018), pp. 102–111. DOI: 10.1109/MNET.2018.1700175.

[264] Sachini Jayasekara, Aaron Harwood and Shanika Karunasekera. 'A utilization model for optimization of checkpoint intervals in distributed stream processing systems'. In: *Future Generation Computer Systems* 110 (2020), pp. 68–79. DOI: 10.1016/j.future.2020.04.019. arXiv: 1911.11915.

[265] Gabriele Russo Russo, Valeria Cardellini, Francesco Lo Presti and Matteo Nardelli. *Towards a security-aware deployment of data streaming applications in fog computing*. Vol. 83. 2021, pp. 355–385. DOI: 10.1007/978-3-030-57328-7_14.

[266] Stefano Forti, Gian Luigi Ferrari and Antonio Brogi. 'Secure Cloud-Edge Deployments, with Trust'. In: *Future Generation Computer Systems* 102 (2020), pp. 775–788. DOI: 10.1016/j.future.2019.08.020. arXiv: 1901.05347.