Coventry University

DOCTOR OF PHILOSOPHY

Application Deployment Framework for large-scale Fog Computing Environment

Verba, Nandor

Award date:
2020

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of this thesis for personal non-commercial research or study
- This thesis cannot be reproduced or quoted extensively from without first obtaining permission from the copyright holder(s)
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 18. Feb. 2021

Application Deployment Framework for large-scale Fog Computing Environments

by

Nándor Verba

September 2018

A thesis submitted in partial fulfilment of the University's requirements for the Degree of Doctor of Philosophy

Certificate of Ethical Approval

Applicant:

Nandor Verba

Project Title:

Application Deployment Framework for large-scale Fog Computing Environments

This is to certify that the above named applicant has completed the Coventry University Ethical Approval process and their project has been confirmed and approved as Low Risk

Date of approval:

21 June 2018

Project Reference Number:

P71986

Acknowledgements

I would like to acknowledge and express my thanks to my director of studies, Professor Kuo-Ming Chao for the long meetings and difficult to answer questions that led to the breakthroughs in my thesis as well as allowed me to have a firm grasp on defending and explaining my work. Furthermore, I would like to thank all other members of my supervisory and research team, Prof. Anne James, Dr. Xiang Fei, Dr. Jacek Lewandowski, Dr. Nazaraf Shah and Dr. Daniel Goldsmith, without whom this work would have not been completed.

I would also like to show my gratitude to members of staff at both the Faculty of Engineering and Computing but also members from the Academic Support Unit. They made my time at the University better and simpler through their help and support.

I am also thankful for the teaching and development opportunities that came with supporting Dr. Norlaily Yaacob and others on their modules. I would also like show my gratitude for the help and support I received from the Manufacturing an Technology Centre, the Future Transportation Centre and also for the funding support from the Global Leaders Program.

Finally, I would like to wholeheartedly thank my family, Helga and my friends for their support during my thesis. They helped me along my way and without them, I could have not finished in time, or at all.

In summary, this thesis could not have been submitted without the support and help of some amazing people and they have my gratitude for all of the things they did to push me and the thesis along.

List of Publications

Journals

[1] **Verba, N.**, Chao, K.M., Lewandowski J.,Shah. N., James, A., Tian F., 2019, **Modelling industry 4.0 based fog computing environments for application analysis and deployment** *Future Generation Computer Systems*,91, pp. 48-60.

- based on Chapter 5 - Application and Gateway Model

[2] **Verba, N.**, Chao, K.M., James, A., Goldsmith, D., Fei, X., Stan, S.D., 2017. **Platform as a service gateway for the Fog of Things.** *Advanced Engineering Informatics*, 33, pp. 243-257.

- based on Chapter 3 - Fog of Things Platform

[3] Fei, X., Shah, N., **Verba, N.**, Chao, K.M., Sanchez-Anguix, V., Lewandowski, J., James, A., Usman, Z., 2019. **CPS data streams analytics based on machine learning for Cloud and Fog Computing: A survey**, *Future Generation Computer Systems*, 90, pp. 435-450.

- used parts in Chapter 2 - Research Background

Conferences

[4] **Verba, N.**, Chao, K.M., James, A., Lewandowski, J., Fei, X., Tsai, C.F., 2017, November. **Graph Analysis of Fog Computing Systems for Industry 4.0**. In *2017 IEEE 14th International Conference on e-Business Engineering (ICEBE)*, pp. 46-53.

- based on Section 6.1 - AME Case Study

[5] **Verba, N.**, Chao, K.M.,Soizic L.,, Eleni A., September. **Smart Transportation platform for big data analytics and interconnectivity**. In *2018 International Conference on Traffic and Transportation Engineering (ICTTE)*, pp. 232-238.

- based on in Section 7 - Future Work

Abstract

The extension of the Cloud to the Edge of the network through Fog Computing can have a significant impact on the reliability and the latency of deployed applications. Recent papers have suggested a shift from Virtual Machines and Container based deployments to a shared environment among applications to better utilise resources. The existing deployment and optimisation methods do not account for application interdependence or the locality of application resources, which can cause inaccurate estimations. When considering models that account for these, however the optimisation task of allocating applications to gateways becomes a difficult problem to solve that requires either model simplifications or tailor-made optimisation methods.

The main contribution of this research is the set of weighted deployments methods that aim to reduce the complexity of the search-space in large-scale fog deployment environments while retaining significant system characteristics. This work was attained by first addressing some existing IoT issues by proposing a Fog of Things gateway platform to answer the connectivity and protocol translation requirements. The proposed platform was used to identify the characteristics and challenges of these systems. A new data-driven reference model was then proposed to estimate the effects of application deployment and migration on these systems. Based on this model, weighted clustering and resource allocation methods are defined, that are then improved upon by a set of weight tuning methods focusing on analysing favourable and sample deployments.

These proposals were validated by running tests on Industry 4.0 case studies. These varying scenarios made it possible to identify the scaling and deployment characteristics of these systems. Based on these initial tests, the second batch of physical and virtual experiments was carried out to validate the models and to evaluate the proposed methods. The findings show that the proposed application and gateway model can predict the load and delay of components to an accuracy of 91%. Within the presented scenarios, constraints and Fog sizes larger than 300 applications, the proposed weighted clustering methods were shown to significantly improve the utility of deployments. In some cases, these methods were the sole providers of valid solutions.

Table of contents

List of figures

List of tables

Nomenclature

Acronyms / Abbreviations

AMQP Advanced Messaging Queue Protocol

API Application Programming Interface

AWS Amazon Web Services

BLE Bluetooth Low Energy

BogoMIPS 'bogus' Million Instructions per Seconds

CGO Costly Global optimisation

CMM Coordinate Measuring Machine

CoAP Constrained Application Protocol

CPS Cyber-Physical Systems

DBSCAN Density-based spatial clustering of applications with noise

FOT Fog of Things

GA Genetic Algorithms

IaaS Infrastructure as a Service

ICT Information and Communication Technology

IoS Internet of Services

IoT Internet of Things

IoV Internet of Vehicles

JSON JavaScript Object Notation

K − Means K-Means Clustering Method

LAP Linear Assignment Problem

M2M Machine to Machine

MQTT Message Queue Transport Telemetry

NaaS Networking as a Service

NP non-deterministic polynomial time

OPTICS Ordering points to identify the clustering structure

OSGI Open Service Gateway Architecture

OSI Open System Interconnection Model

PaaS Platform as a Service

PSO Particle Swarm Optimisation

QAP Quadratic Allocation Problem

QoS Quality of Service

REST Representational State Transfer

RFID Radio Frequency Identification

RPC Remote Procedure Call

SAaaS Sensing and Actuating as a Service

SaaS Software as a Service

SDN Software Defined Networking

SenML Sensor Markup Language

SLA Service Level Agreement

SOA Service Oriented Architecture

SOC System on Chip

SOM Service Oriented Manufacturing

$STOMP$ Stream Text Oriented Messaging Protocol

$TaaS$ Things as a Service

URL Uniform Resource Locator

VM Virtual Machine

WAN Wide Area Network

WSN Wireless Sensor Networks

Methods / Functions

$R(t)$ Reliability in function of time t

$z(t)$ Hazard or Failure Rate Model at time t

$z(x)$ Hazard or Failure Rate Model at Load x

Variables

A_{ij} Application i deployed on Gateway j

$Alloc_{BaseToULoad}$ Base Load to Unit Load Ratio for Gateway Allocation

$Alloc_{Capab}$ Capability similarity ratio for Gateway Allocation

$Alloc^{i}_{Clust}$ Resource Share Parameter for Gateway Allocation for Clusters

$Alloc_{PerfCapToULoad}$ Performance Capability to Unit Load ratio for Gateway Allocation

$Alloc_{ResShare}$ Resource Share Parameter for Gateway Allocation

$Alloc_{SpeedToULoad}$ Performance Speedup to Unit Load ratio for Gateway Allocation

$Alloc_{ResShare}$ Resource Share Parameter for Gateway Allocation

AVG_{Route} Average path length in the Fog

CCF Clustering Coefficient

C^{i}_{App} Cluster area i for the Application

Ct^{F} Total Constraint Violations in the Fog

$W_i^{ConstraintViolations}$ Weight of Constraint Violations for Application i

Cls_{Dist} Distance Parameter for Clustering

$Cls_{MsgRate}$ Message Rate Similarity Parameter for Clustering

Cls_{ReqSim} Requirements Similarity Parameter for Clustering

Cls_{Constr} Constraint Similarity Parameter for Clustering

$Cls_{Cls_{Sum}}$ App to App Total Distance for Clustering

Cls_{ULoad} Unit Load Similarity Parameter for Clustering

Cls_{UtilW} Utility Weights Similarity Parameter for Clustering

D_{ij}^A Total Delay of Application i on Gateway j

D_{iRef}^A Reference Delay for Application i

D^F Total delay in the Fog

$D_{j,k}^{Ping}$ Total delay from Gateway j to Gateway k

$Diameter$ The Diameter of the Fog

D_{ij}^N Networking Delay of Application i on Gateway j

D_{ij}^P Processing Delay of Application i on Gateway j

$D_{j,k}^{Ping}$ Ping Value between Gateway i and j

D_{Base}^R The Base Networking Delay on the Platform

D_{Ext}^R External Jump Routing Delay

D_{Tot} Total Application Delay on the System

eps minimum distance for two nodes to be Neighbours

GBC Graph Betweenness Centrality

GDD Graph Degree Distribution

G_j Gateway with id j

$k1, k2$ Time Delay Calculation constant

L_{ij}^{A} Application Load on Gateway j on App i

L_{ij}^{a} Measured Application Load on Gateway j on App i

λ_{ij}^{A} Message Rate of Application i

λ Constant Failure Rate

λ_{jk}^{Gw} Message rate of Driver k on Gateway j

L_{j}^{B} Base Load on Gateway j

L_{k}^{D} The Load of Driver k

L_{j}^{Gw} Load on Gateway j

L_{j}^{Idle} Idlea Load on Gateway j

LM Message Load

Loc_{k}^{i} The Locality of type i of application k

L_{ij}^{u} Unit Load of Application i

$minPts$ Minimum number of points to form a Cluster

$Util^{F}$ Estimated Utility in the Fog

P_{j}^{Cap} Processing Capacity of Gateway j

P_{j}^{Speed} Processing speedup of Gateway j

P_{Ref}^{Speed} Reference Speedup Value

R_{ij}^{A} Reliability of App i

R^{F} Total Reliability of the Fog

R_{j}^{Gw} Reliability of Gateway j

R_{Ref} Reference Reliability for the system

R_{kl}^{Res} Reliability of Resource k on Gateway l

R_{App}^{Type} Resource use of Application

r_xy Correlation between parameters x and y

S_x Standard deviation for parameter x

T_j^{Gw} Execution time of process on Gateway j

T_{Ref}^{Gw} Reference execution time of process

$Util_i^C$ Utility of Cluster i

$\overline{Util_i^C}$ Estimated Utility of Cluster i

$Util_{Delay}^F$ Utility value for the Delay Optimisation scenario

$Util^F$ Total utility of the Fog

$Util_{Multi-Comp}^F$ Utility value for the Weighted Multi-Component Optimisation Scenario

W_i^{Delay} Weight of Delay for Application i

$W_i^{Reliability}$ Weight of Reliability for Application i

$\overline{x}$ Mean values of parameter x

Chapter 1

Introduction

1.1 Research Context

The concepts of Industry 4.0 provide a new means of integrating concepts from ubiquitous computing with manufacturing technologies through cybernetics. This advances the automation of the manufacturing systems and helps improve product quality, production efficiency, condition monitoring and decision making (J. Lee, Bagheri, and Kao 2015; DIN 2016). Within this concept, machines become connected with humans through computer systems to work in a coordinated way to automate data acquisition, sharing and exchange among the physical and virtual worlds.

The wide spread availability and affordability of sensors, wireless networks and the accessibility of high-speed Internet make real-time multiple parameters monitoring and control of manufacturing process possible in a way that was not feasible before (Y. Lu 2017). This leads to a great number of sensors being deployed to physical machines which in turn generates a large volume of data that requires computationally intensive analysis and interpretation for decision-making purposes. The resulting decisions, whether made by humans or software, often need to be transformed into control signals for actuators to operate the machine in the physical world. This then creates a loop-back to the sensor system as new sets of data are collected and sent back for further analysis, reflecting changing machine states over time.

This type of system based on Cyber-Physical System (CPS) is a facilitator for realising the concepts of Industry 4.0. It enables computational algorithms and physical components to interact with each other through real-time monitoring and control to improve productivity (Trappey et al. 2016; L. Wang, Törngren, and Onori 2015). Yet, as stated in (Wiesner, Marilungo, and Thoben 2017) traditional servers with limited capacities may not be able to cope with the new challenges in terms of scalability and complexity of such systems. In turn,

the Cloud with Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) provides a promising integrated solution (Mell and Grance 2011; Chaâri et al. 2016) to address these challenges.

However, the cost of using Cloud services and the latency between the edges of the network and the Cloud could hinder its application to time critical applications. Furthermore, due to health and safety issues, some machines cannot be operated remotely and can only be operated within particular boundaries at the premises (Health and Safety Executive 2004). Besides, in some cases, the control software and the machine are bundled together due to security, license and driver requirements, so it cannot be run on the Cloud or other machines. On the other hand, the computational power or other resources on the controllers may have spare capacity to process additional tasks. The study of assigning these existing extra resources to time-sensitive tasks and overloaded computer systems becomes an interesting alternative to Cloud processing (Deng et al. 2016; Jiang 2016). Such systems, as part of a network infrastructure, could efficiently deal with front-line demands (Bonomi et al. 2012a).

The Fog environment is an extension of the traditional Cloud to the edge of the network including both Cloud and edge resources. In Cloud computing environments, the physical machines are considered to have mostly homogeneous builds with network latencies inside clusters small enough that the deployed location of applications is not considered. In Fog environments, locality is considered to be an important issue, due to the heterogeneous build of nodes, the higher communication latencies inside a wireless network and the physical lock-in of device resources. These differences between Fog and Cloud approaches introduce the need for a new workload descriptor for a PaaS based Fog. Furthermore, in CPS systems the importance of device to application communication needs to be considered. These require the formulation of a new Application and Gateway Model as well as new optimisation approaches that can address the differences between these systems.

1.2 Research Problem

So far, only a limited number of optimisation and load balancing approaches have been proposed in the context of Fog computing and CPS (Do et al. 2015). The existing approaches (Mahmud and Buyya 2016), however, focus on the design of optimisation methods using different simulated parameters such as workload, power consumption and virtual machine distribution. These approaches also lack physical environment testing, only considering theoretical formulas for delay and processing that can only be implemented with full knowledge of the applications and the systems that is not always possible. There are few studies (Díaz, Martín, and Rubio 2016) on application properties such as coupling nature between

devices/drivers and controller/gateway. Such properties can lead to extra communication delays, depending on the distance between devices and gateways. The existing approaches (Mahmud and Buyya 2016) are mainly interested in the overall or average workload balancing without considering individual message response time, which can be crucial for CPS.

Based on these deficiencies in the current state of the art and based on the possible prospects of future industrial requirements the main research question can be summarised as:

- **How can large application systems deployments be analysed and improved in highly heterogeneous Fog environments?**

To be able to answer the main research question, a broad set of questions need to be defined. These look at defining the challenges involved in answering the main question and at defining what should be improved and analysed on the system. These questions are as follows:

- **What are the requirements and characteristics of future Industry 4.0 Gateways and how can these be translated into protocols and systems?**
- **How can changes in the model be analysed and estimated using the run-time parameters and connections of the applications and gateways?**
- **What are the challenges of application deployment in Fog systems and what methods can be proposed to diminish their effects?**

These questions were used when formulating the research aims and objectives in the next subsection and form the basis of the validation and the chapter orchestration as well.

1.3 Research Aims and Objectives

The research aim can be defined as the broad challenge of this work and is stated below.

Formulate, implement and evaluate an IoT and Fog based Application Deployment Framework for Industry 4.0 systems

The objectives can be described as steps that need to be taken to fully answer both the main research question and the more precise questions that make it up. Due to the nature of the work the objectives can be split up into three categories, Platform, Model and Method. Each of these looks at distinct components of the big framework.

Platform

- Provide a high-level framework for an IoT based Fog Platform that can satisfy the requirements presented by the current State of the Art.
- Implement the framework using existing technologies and protocols.

Model

- Analyse the Platform, identify relevant parameters and determine their behaviour.
- Formulate the Platform model taking into account numerous heterogeneity components.
- Identify system specific behaviours and constants, validating these.

Method

- Analyse the optimisation problem determining its hardness and propose directions for improvement.
- Formulate methods that take advantage of system characteristics to meaningfully reduce the search space for the optimisation methods.
- Evaluate proposed methods to identify advantages and disadvantages together with desired use-case scenarios.

The objectives are translated into upcoming sections of the thesis where each set of objectives is grouped into a chapter and validated in Chapter 7.

1.4 Methodology

1.4.1 Research Methodology

The research methodologies used when conducting this research changed based on the alteration or iteration of the research questions as components were implemented and new information was found. In overview, a collection of mixed methods was used in such a case as suggested in (Borrego, Douglas, and Amelink 2009). Quantitative methods were used to evaluate the benefits or drawbacks of certain approaches while qualitative methods and data gathering were used to determine why certain methods might perform better than others.

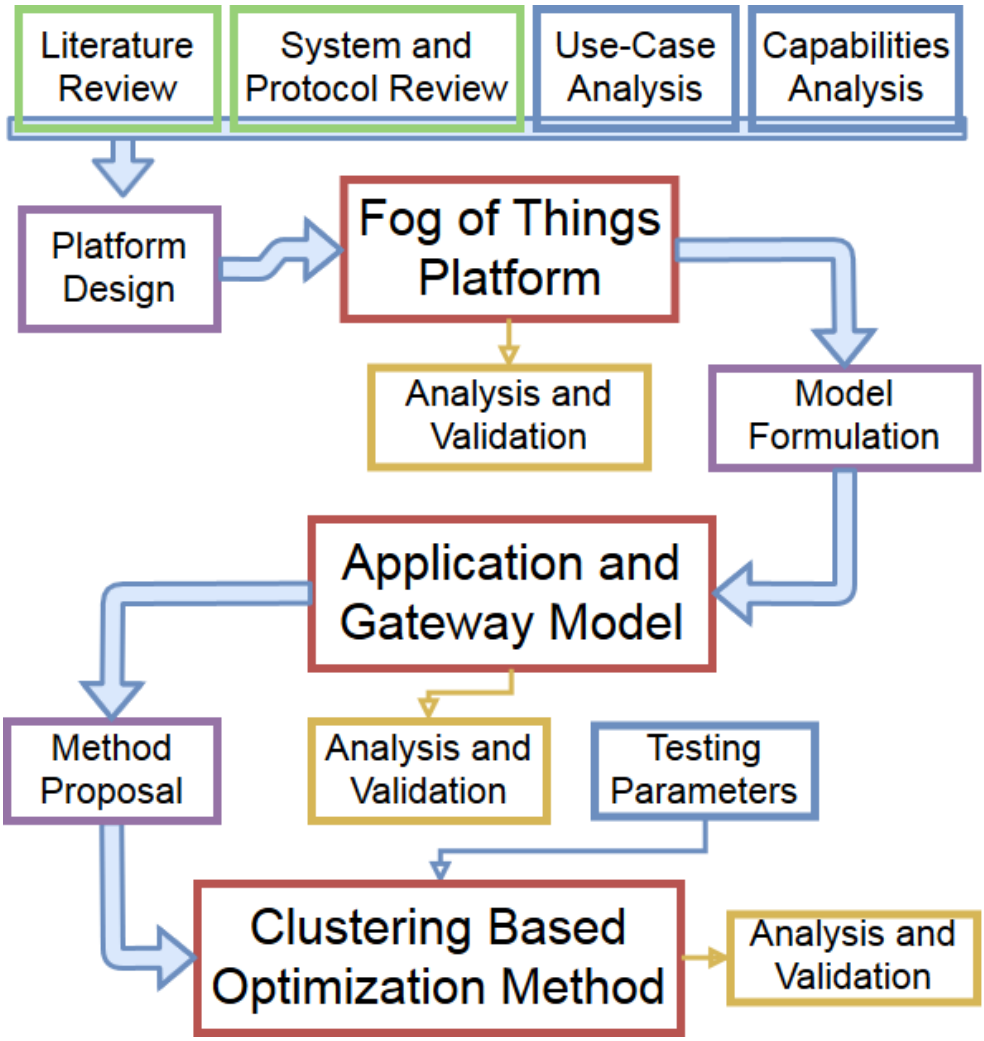

Fig. 1.1 Overview of Research Methodology

Qualitative methods were used in the evaluation of the platform where its adherence to the Fog and IoT requirements was examined.

The overview of the research methods can be seen in Fig. 1.1 where the three stages can be seen and the components of these are described on how they contributed to the final result. The three stages of the research can be broken down to the three main outcomes that are the Platform, the Model and then the optimisation Methods proven on these.

1.4.2 Fog of Things Platform

The first component was designed to answer the requirements of Industry 4.0 and also to explore some of the novel concepts of IoT and Fog Computing. To decide which components are to be used on the platform a similar methodology was used as in (Cruz et al. 2018) where

a detailed literature review was done to identify direction and requirements while a Systems, platform and protocol review was done to see which is mature enough and has the right functionality to answer the requirements. Furthermore, a set of use-cases were considered with requirements that the platform needs to answer. A final component that was considered when formulating the concept of the platform was the capabilities of the physical devices and team members.

Based on these stages the Platform design or framework stage was undertaken that had three distinct components. Requirements were formulated for the platform as well as a set of expected improvements to the current state or to a reference state. Based on these requirements, components and technologies were chosen from the tested set and the Platform was implemented. After the implementation, the components were validated to show they correspond to the requirements and a run-time analysis was done to show how certain parameters change on the system and how these are in-line with the expected improvements.

1.4.3 Application and Gateway Model

The application and gateway model was formulated after the analysis and validation of the platform. In the first instance, a niche literature review was done where future directions were analysed and the state of the art regarding application models for deployments, drawbacks and shortcomings were identified as well as opportunities for improvement. With identifying the parameters, that looked at which components of a deployment can be measured using the platform and how these can be related to the ones that cannot but are crucial in such systems.

The method of identifying and using the models found is similar to the one used in (Sargent 2007) where the formal model was designed, and then it went through a number of iterative stages of altering and modifications until it could estimate changes and deployments to match the requirements.

To validate the model there were three sets of tests that were performed. Initially, a set of tests was run to identify the parameters of a system while afterwards the model was validated using these parameters in two sets of tests. The first test looked at individuals being deployed in controlled environments and how their behaviour changed, while the second one looked at more bundled and complex deployments where the overall estimation capacity was analysed.

1.4.4 Clustering based optimisation method

The optimisation methods that could be applied to a system are largely reliant on the model that is being used as shown in (Aibinu et al. 2016). To determine what method types can employed, another niche literature review was performed looking at what methods were

proposed to allocate applications to gateways. Based on this review, and analysis of the characteristics of the model and a Hardness test to analyse how difficult the problem is, the optimisation method and its components were formulated. A number of possible directions and solutions were found and analysed and as in the previous section in an iterative fashion they were improved or replaced.

1.4.5 Validation and Analysis

When considering the validation of the optimisation method the guidelines of (Brownlee et al. 2007) were followed where the methods were restricted to a certain set of use-cases and testing data variation, limiting their generality but which also allowed the components to be thoroughly tested and a qualitative view of the results to be available rather than just a quantitative one. The scaling model and generated test cases were based on the industrial use-cases and their graph-based analysis to verify how components interact and what type of scalability model they follow.

The optimisation methods were analysed in three distinct ways. The first one looked at evaluating the effect of each component on the results of the system, attempting to reduce the impact of variability so the singular effects can be seen. The second component looks at comparing the proposed methods to some existing ones, or slight variations of these to see how they perform and what is the trade-off between these as the no free lunch theorem (Wolpert and Macready 1997) suggests. The third set of evaluation tests look at determining how the proposed methods work in scalability scenarios. This section looks at complexity analysis as well as the increasing difficulty of finding valid solutions as large sizes are reached.

1.5 Novelty and Contribution

The main contribution of this thesis is the weighted clustering methods that aim to solve the scaling challenges of future large-scale Fog deployments that were based on the identified characteristics, models and challenges of these systems.

When considering the Fog of Things Platform, its main contribution can be summed up by the formulation of the Fog of Things paradigm and its implementation or architecture proposal based on existing and modified protocols and systems. The paradigm shift can be summed up by considering connected devices as resources and the attempt to create a homogeneous application environment on top of a highly heterogeneous system.

The novelty and contribution of the Application and Gateway model lie in its data and system driven formulation where only measurable parameters are considered and the full heterogeneity of the system is taken into account. This results in a model that can be deployed in existing systems and can make more accurate predictions.

Within this framework, each component has novel elements, but the deployment optimisation method provides the most novelty with the highest impact. This component focuses on the creation of clusters to meaningfully reduce the search-space in the system. The weighted property-based clustering and resource allocation, as well as the identification and training of these weights is at the centre of this work.

1.6 Thesis Structure

The thesis structure follows the structure of the objectives and the research questions. Each chapter is dedicated to answering a set of objectives and their questions. These chapters are preceded by a background and overview information chapter and followed by their evaluation and the conclusions of the research.

The thesis is structured in a logical way based on the proposal, where the whole thing can be considered a framework which includes a Platform, a Model and an Optimisation method. This structure is followed in the Research Background section and throughout the thesis and in the Evaluation as well.

The Fog of Things Platform, the Graph Analysis of the System and the proposed Gateway and Application models have been published by this research team in the course of the ongoing research for the PhD. These have been linked in the Published Papers section and certain paragraphs and sections from these were included in the Literature review, Introduction, Body and Evaluation chapters.

Chapter 1 - Introduction aims to give a general introduction to the thesis. It also contains a highlight of the major contributions, methodology, components and research questions.

Chapter 2 - Research Background gives an overview of the existing state of the art, potential directions, aspects of solving each problem and the work these are based on.

Chapter 3 - Fog of Things Platform describes the proposed platform, its components and the theoretical and practical aspect of implementing these. This chapter discusses characteristics and the shift in paradigms resulting from the inclusion of Things in the Fog.

Chapter 4 - Application and Gateway Model formulates the model for Application and Gateway delay, load and reliability calculation that support the system health and the

utility calculations. This is built on the previous chapter and provides the foundations for the optimisation methods described in the next chapter.

Chapter 5 - Deployment Optimisation showcases the components of the optimisation methods as well as variants of it. In this section, these components are described together with the four main configurations that are being tested. This chapter builds on the previous two and provides the main contribution of the thesis.

Chapter 6 - Evaluation and Analysis evaluates and analyses the components, models and the platform presented in the previous three chapters. Its purpose is to provide a systematic review of the proposed ideas and show their drawbacks and advantages as well as providing a typical use case for these.

Chapter 7 - Conclusions and Future Work summarises the whole thesis and its findings, looking back at the initial questions aims and objectives, how these were answered and the main finding of this research.

Chapter 2

Research Background

2.1 Internet of Things

The concept of ubiquitous sensing and computing is slowly becoming a reality with enabling technologies like Wireless Sensor Networks (WSN) and the widespread adaptation of Cloud computing. These advancements, as well as innovations in the fields of Radio Frequency Identification (RFID), advancements in Wireless technologies like 4G and the introduction of low powered microelectronics that enable wireless communication, sensing and actuation control in microchips, and System on Chip (SOC) devices have contributed to the emerging field of Internet of Things (IOT).

The realisation of the IoT would represent the advancement to Web3 (ubiquitous web) as described by (Gubbi et al. 2013), where Smart Objects and Devices will be seamlessly embedded in their surroundings. Implementation of such a concept gives rise to a new set of challenges, such as uniquely identifying Objects in the system, gathering, interpreting, storing and visualising the data, and being able to manage security, faults and billing requirements for such devices.

There are numerous visions for a functioning framework for the IoT. These can be split into three main categories: Thing Oriented, Internet or Middleware Oriented, and Semantic Oriented. A generic overview example can be seen in (Hakiri et al. 2015). Each of these approaches has their own advantages and disadvantages. These categories focus on categorising IoT systems based on their architecture. Some systems might incorporate ideas from two or more of these categories to accomplish their objectives.

A Thing Oriented approach can be seen in (Gubbi et al. 2013) where the authors give a definition to the IoT as well as define its requirements, together with use cases and proposed architecture on both public and private Clouds. This approach envisions objects that are connected to a network that can interact with their environments through sensing and

Some materials have been removed from this thesis due to Third Party Copyright. Pages where material has been removed are clearly marked in the electronic version. The unabridged version of the thesis can be viewed at the Lanchester Library, Coventry University.

Fig. 2.1 (Hakiri et al. 2015) Overview of IoT Vision and Architecture

actuation as well as communicate with peers and analyse data. The authors see this step as a "move from www (static pages web) to web2 (social networking web) to Web3 (ubiquitous web)". A Semantic Oriented approach is discussed in (Gyrard et al. 2015) where the authors define the requirements of such an approach as well as analysing current contributions to the field while defining their limitations and constructing a new architecture. This approach is a data-oriented one, where the focus is on the abstraction of tasks with an emphasis on communications and the transfer of data and interpretation of information.

A Middleware Oriented approach can be observed in (Distefano, Merlino, and Puliafito 2015) and (Sarkar et al. 2015) where the main focus is on the layers and architectures that control the devices using different virtualisation policies to make the devices available on the network. In (Distefano, Merlino, and Puliafito 2015) they envision a system where the "lower level functionalities" are separate from the network which would be a Software Defined Network (SDN). In (Sarkar et al. 2015) they define a layered design that adds a virtualisation layer on each level that connects the devices with the application.

2.2 Fog and Cloud Computing

The interconnection of sensor and actuator systems with decision making and analytics have traditionally been performed by either local static controllers or sent up to the Cloud for analysis. Through the paradigms of Internet of Things (IoT), Cloud computing based systems propose the virtualisation of devices and provide their data and connection as a service for users within a Sensing and Actuation as a Service (SAaaS) as proposed in (Distefano, Merlino, and Puliafito 2015) or Things as a Service (TaaS) in (Christophe et al. 2011). Another role Cloud computing has in CPS is focused on the analytics of the data received from devices. The Cloud can provide a vast amount of processing and storage resource (Fox, Kamburugamuve, and Hartman 2012) which can be used to analyse large amounts of data (Zhang et al. 2017) or streams (Hossain et al. 2012). These Cloud capabilities are focused in data centres (Rui and Danpeng 2015) which are centralised and have a remote nature, which has several drawbacks. The security aspect of storing, analysing and managing data in the Cloud is an increasing concern (Botta et al. 2016), while the remote nature of the Cloud also has reliability and latency issues (Stojmenovic 2014).

The paradigms of Fog Computing as proposed by CISCO in (Bonomi et al. 2012b) extend the Cloud to the edge of the network to better utilise resources available on gateways and devices connected to the network (Cisco Systems 2016). This extension allows data to be stored and processed locally increasing reliability and security while decreasing the latencies between devices and the processing elements (Dastjerdi and Buyya 2016). The hosts or gateways used in Fog systems vary from PC based Computing Nodes (Aazam and Huh 2014), Mobile Devices (Hong et al. 2013) to resource-constrained System on Chip Devices (SoC) (Jalali et al. 2016). These hosts all have varying storage, processing and networking capabilities (Giurgiu et al. 2009). Intel NUC and other small form factor style compute nodes are the most common and are the most reliable, but these can usually only communicate with devices using Ethernet or WiFi-based networks. An example of such a system can be seen in (P. Verma and Sood 2018). The mobile device and SoC-based devices have fewer resources but provide a wider range of wireless communication possibilities for a polyglot gateway (Datta, Bonnet, and Nikaein 2014), such that they can be used to connect to a wider range of heterogeneous devices that can use low-power Machine to Machine (M2M) communication protocols.

The platforms deployed in Fog computing vary based on hosts and application domains, but they can be categorised in a similar way as in Cloud computing. Infrastructure-based solutions allow users to deploy Virtual Machines (VMs) (Luiz Fernando Bittencourt et al. 2015) or Docker Images (Bellavista and Zanni 2017). Platform-based solutions as in (Al-Fuqaha et al. 2015; Khodadadi, Calheiros, and Buyya 2015; Paraiso et al. 2012) provide a

Some materials have been removed from this thesis due to Third Party Copyright. Pages where material has been removed are clearly marked in the electronic version. The unabridged version of the thesis can be viewed at the Lanchester Library, Coventry University.

Fig. 2.2 (P. Verma and Sood 2018) Example of Fog Environment

platform for users for application style system deployments. The third type of solution as shown in (Gyrard et al. 2015; Z. Li 2016) provides networking and analytics capabilities that the user can only configure and use without the need to program and deploy their own application or platform. Some Cloud solutions also focus on the interconnection and management of these services as in (Kum et al. 2015)

From the hosts perspective, there are a number of differences between Cloud and Fog. The main difference between these systems is the amount of resources they possess, while Cloud Systems are considered to have virtually unlimited storage and processing capabilities (Aazam, Khan, et al. 2014), Fog systems are a lot more restricted, making their efficient management more crucial (D. Kim, C. Lee, and Helal 2015). When looking at inter-host communication in the Cloud, due to high-speed networks these delays are generally considered to be uniform and negligible. In the Fog systems, due to wireless communication and varying network types, networking delays can vary largely between hosts (Gupta and Garg 2015). When looking at the device to host communication, the Fog is closer to the remote devices while the Cloud adds significant networking delays when accessing them. When looking at the differences from a platform perspective, the Cloud solutions offer full

control of resources using VMs, Docker style solutions or other Platform as a Service (PaaS) options. Fog solutions tend to share resources between different users and systems.

Cloud-based CPS and IoT systems are typically designed using smart internet enabled devices that connect to Cloud services using either Message Queue Transport Telemetry (MQTT) (Truong and Dustdar 2015; Singh et al. 2015) or Constrained Application Protocol (CoAP) (Kovatsch, Lanter, and Duquennoy 2012). These devices have higher power consumption and, due to the security and protocol requirements, require more resources to run than their Machine to Machine (M2M) technology based counterparts. In Fog systems these smart devices are replaced by gateways that have significantly more resources and use more power, but they are connected to devices using, Bluetooth Low Energy (BLE), Zigbee, or other M2M specific protocols (Rahmani et al. 2015; W. Lee et al. 2016) that are more power efficient. Hybrid systems as in (Jayaraman et al. 2014) suggest moving real-time sensitive components closer to the edge or in the Fog while leaving resource intensive processes in the Cloud. This can be seen with data-stream processing as well in (Baccarelli et al. 2016) where initial analysis is performed on gateways and the data is sent up to the Cloud for more in-depth analysis. In all of these cases, the decision to move from Cloud to Fog is based on design choices that are influenced by processing requirements, distributivity of data and the real-time requirements of the system (Stojmenovic 2014).

2.3 Industry 4.0 Requirements

In recent years there has been an increased interest in the development of manufacturing systems that allow the monitoring and control of certain parameters, which can respond to faults, be reconfigured, easily deployed and upgraded. Some of the main features and requirements of such systems are presented in (Lasi et al. 2014). The research in this field has resulted in a number of directions such as Cloud Manufacturing, IoT logistics, Smart Manufacturing and others that have at the core the automation of manufacturing tasks, the synchronisation of device usage, and the diverting of more of the management and organisation tasks from humans to a management system. An overview was presented in (Barreto, Amaral, and T. Pereira 2017).

There are multiple challenges and directions that can be taken to create a fully Automated Manufacturing Environment. One of the components is the orchestration of the resources that are needed to make products, reducing Time to Market, manufacturing times and idle devices and resources. A Cloud computing and Enterprise Model based solution is presented in (Bi, Da Xu, and Chengen Wang 2014) that focuses on the management of resource components

Some materials have been removed from this thesis due to Third Party Copyright. Pages where material has been removed are clearly marked in the electronic version. The unabridged version of the thesis can be viewed at the Lanchester Library, Coventry University.

Fig. 2.3 (Barreto, Amaral, and T. Pereira 2017) Industry 4.0 Use-Case

of Cloud-based manufacturing, allowing delivery and deployment models to be implemented based on known characteristics.

Other research (Bi, Da Xu, and Chengen Wang 2014) focuses on the technology component of the system that allows devices from different mediums to send information in a heterogeneous system. The system is broken down into multiple levels abstracting unneeded information at each level, while ensuring the mapping and access to resources through manufacturing services in a Service-Oriented Architecture(SOA) system, thereby offering a Service Oriented Manufacturing (SOM) solution.The platform presented in (Tao et al. 2014) suggests the collaboration of a number of higher-level systems such as the IoTs for resource management and orchestration, the Internet of Users for requests and tasks and the Internet of Services (IoS) for Cloud resource utilisation.

The proposed gateway platform would perform the tasks of the IoT component while overflowing into the IoS region allowing the use of local resources to increase the configurability and Quality of Service (QoS) specifications of manufacturing systems. Furthermore, from an industrial management point of view the gateway platform would allow the fast reconfiguration of systems to decrease the time to market and allow for higher flexibility in industrial production.

2.4 Gateway and Middleware Platforms

IoT gateways have become increasingly configurable and their functionalities have expanded. The horizontal integration directives aim at allowing platforms and devices from different providers using different protocols to interact. This would increase re-usability and reduce application complexity. To allow the connection of multiple devices multi-M2M protocol support, registration, management and enhanced configurability is needed for these devices. The increased number of resources available on the gateway has led to the need to be able to virtualise these and move a portion of the resource use from the Cloud to the edge of the network.

When discussing the IoT, three distinct approaches for the architectures of such systems (Fortino et al. 2014b) can be considered. An IPv6 based network where Smart devices are uniquely accessible through Constrained Application Protocol (CoAP) or other lightweight protocols as has been suggested in (P. P. Pereira et al. 2013). In the Cloud-oriented approach devices are accessed from the Cloud through API's (Fox, Kamburugamuve, and Hartman 2012) or using Message Queue Telemetry Transport (MQTT) protocol. In the middleware approach, information goes through gateways or brokers that communicate with devices through more lightweight communication protocols such as 6LoWPAN, nRF24L01 or ZigBee and forward these messages to the Cloud or local users such as in (Sarkar et al. 2015).

With the introduction of Cloud and Fog Computing paradigms, the use of resources available at the edge of the network is considered as well as the deployment of application and processing tasks on the edge devices (Chao et al. 2015). Proposals like MADCAT in (Inzinger et al. 2014; Kovatsch, Hassan, and Mayer 2015) suggest large applications be decomposed into components and deployed onto devices while (Khodadadi, Calheiros, and Buyya 2015) suggests a MapReduce like approach with IoT application development. Together with proposals from (Ruckebusch et al. 2016) which looks at reconfigurable components and (Fortino et al. 2014a) which looks at agent-based cooperative smart objects, these suggest a need for Software Defined Networking as well as the need of decomposing applications into components and running them on the gateway.

The idea for the use of resources on the edge of the network was first introduced by Cisco (Bonomi et al. 2012b). Advances in Networking as a Service (NaaS) and the increased processing power of gateway devices have led to the development of edge computing platforms like Docker (Ismail et al. 2015). Edge computing includes solutions based on Virtual Machines (Vogler, J. M. Schleicher, et al. 2015; Vögler et al. 2016) as well as container-based application deployment solutions.

Open Service Gateway Interface (OSGI) is a modular service platform for Java that has been the focus of research towards modular IoT Gateways. OSGI can be used for multi-tenant

Cloud connection architecture as in (Azeez et al. 2010). Platforms like HEPA (Seo et al. 2015) propose the use of Zookeeper to control a set of OSGI Gateways that would facilitate the transmission and translation of device information. One of the drawbacks of the OSGI core platform is that it lacks solutions for asynchronous communication between components. To address this issue (Koschel et al. 2012) and (Sivieri, Mottola, and Cugola 2016) proposed a messaging-based solution that maps messages to either internal services or to an event administration system component.

There are a number of different approaches to the design and implementation of IoT gateways. Most of the initial approaches as well as some of the latest ones like Eclipse Scada, Krikkit, SmartHome and HePA (Seo et al. 2015) concentrate on semantic interpretation of data and configuration based routing or event creation. Other approaches like that of Eclipse Kura (*Eclipse Kura* n.d.) and Eliot (Sivieri, Mottola, and Cugola 2016) look at fully re-configurable systems where applications configure and define everything, a fully modular system. These approaches cause platform and provider lock-in where information passing between peers is problematic. Solutions like BUTLER (Botella et al. 2009) and the use of eTrice provide an abstraction of protocols to enable easier application development. While eTrice (M. H. Orabi, A. H. Orabi, and Lethbridge 2016) generates Java or C code based on the written code, Butler deploys the run-time environment directly onto devices, which allows the user to review the written code. Most of the presented gateways have very limited solutions for re-configuring and re-programming connected devices to suit the needs of the users. However, there are proposals like GITAR (Ruckebusch et al. 2016) and MADCAT (Inzinger et al. 2014) that provide a platform that can reconfigure embedded devices connected to it. A functionality based summary review can be seen in Table 2.1 where the differences between the gateway platforms are highlighted based on six most commonly considered criteria.

Due to the differences in processing and storage resources of IoT devices there is a wide range of tailored M2M protocols. The higher level protocols like CoAP, SNMP and MQTT-SN are used on devices that have higher processing and power resources available. More resource constrained devices use protocols with lower levels of abstraction and functionality, such as 6LoWPAN, XBee, RF24 or even core 434 MHz, each having their preferred implementation scenarios and varying advantages and disadvantages. This protocol fragmentation has led to increased research regarding the brokering and semantic translation of received messages from the existing protocols such as in (Al-Fuqaha et al. 2015). Architectures like Krikkit and BUTLER look at mapping to REST requests with notification feedback. In contrast, BUTLER attempts the handling of asynchronous requirements of IoT Systems with the use of a messaging service that provides this implicitly. The solutions like Kura and

Table 2.1 Platform and Middleware Features

Gateway Platform	Horizontal Integration	Multi M2M protocol	Multi Cloud Tenancy	Deployable App Layer	Protocol Agnostic Messaging	Local Resource Use*
Krikkit	X	X	–	–	X	R
SCADA	–	X	–	–	–	R,S
Kura	–	X	–	X	X	P,N,S,O
SmartHome	–	X	–	–	–	P,R
eTrice	X	X	X	X	–	P,R
HePA	X	X	X	–	X	P,N
BUTLER	X	X	X	X	X	P,N
GITAR	X	–	–	X	X	P,N
ELIoT	–	–	X	–	X	P,N,S

*P-Processing R- Message Routing, N- Full Networking, S-Storage, O-Other

ELIoT use MQTT as a messaging service with the Cloud and translate all device messages to MQTT. The drawback of some of the presented solutions is that while they offer a uniform and configurable communication means with the Cloud, they do not provide a protocol agnostic message passing system for applications and device messaging.

The management and northbound or Cloud oriented connections of the gateways have a number of approaches that can be used. Gateways like Krikkit, Eclipse SCADA, Kura, SmartHome and BUTLER all use RESTful APIs and User Interfaces for control and management. Platforms like Kura and Krikkit allow for MQTT Cloud connections to be configured for message passing. BUTLER allows for multiple Cloud tenancies through connections made through the REST APIs which can connect to local area and Cloud resources as well as other smart devices. Approaches like HePA suggest proxying through CoAP for passing of control and device data between gateways. While these approaches allow for some basic networking configuration they lack a truly software defined networking platform that would support the configuration of multiple networking connections that not only allow message passing but management and application deployment as well.

The requirements for the horizontal integration of devices has become evident in the past years with an increased amount of platforms switching from a vertical view, where platforms have their specific protocol and device support, to a more horizontal one, encapsulating different protocols and device connections from other providers. This is leading to an increased interconnectivity and the use of multi-tenancy connections for features available from different providers. Most of the presented platforms like Kura only allow one MQTT connection to be configured. While applications can implement the drivers and have their

own connection, this is not implicit to the platform. BUTLER provides the most extensive support for this, supporting multiple types of devices like smart phones, local computers, gateways and Cloud connections. In general however, these platforms provide a mostly vertical view of the system with connected devices and messages still being confined to their respective gateways and these needing to be updated and deployed independently. There is a requirement for a more loosely coupled connection among devices, applications and resources, where these applications can pass messages seamlessly from different containers to devices connected to other available gateways without needing to be rewritten. This would allow for functionalities like migration, clustering and high availability to be exploited on these gateways which could lead to higher quality of service (QoS) standards.

The use of resources at the edge of the network is one of the main concerns of Fog Computing as described by Cisco (Bonomi et al. 2012b), with the use of processing and networking resources being of main concern. Some platforms suggest deploying VMs such as in ELIoT or allowing the users to configure the data processing as with the Krikkit, SCADA and SmartHome platforms. Solutions like Kura, eTrice and BUTLER suggest the deployment of applications onto these gateways which allows for faster deployment and more efficient use of resources but constrains the users to platforms or languages whereas VMs allow for full control of the environment. Although the storage resources available on the gateways are rarely discussed, there is research where the use of software like CouchDB and PouchDB for Fog computing devices is evaluated (Kimak and Ellman 2013). In these scenarios, device related information is stored locally and updated with the Cloud when needed. Context resources are made available to applications which may include location as in Kura or region information as made available in HePA. A more comprehensive view on how to manage these resources is needed as well as a need to be able to combine resources management systems from different languages and platforms.

Based on the review of the existing platforms as well as the direction of the IoT community, it can be concluded that there is a need for more horizontal integration of gateways as well as a need for a protocol agnostic messaging system for applications to talk to devices. Furthermore, in evaluating the current platforms it is obvious, that certain aspects have received a lot of attention and have had good solutions, especially in the case of BUTLER, but there is still a room for improvement. The resource availability and use by gateway applications, as well as in the creation of protocol agnostic and event based device messaging environment for the applications, and not just the Cloud are key issues for the development of future platforms. In addition to the platform lock-in created by a vertical, single platform approach for most of the existing systems, device and protocol dependent solutions create a

big impediment for application development of devices from multiple providers and protocols that provide similar functionalities, reducing the capabilities and reusability of such systems.

If gateways are to cope with the proposed interconnectivity and the wide range of devices, use cases, protocols and QoS requirements of future environments, they need to be able to connect to multiple Cloud providers that may offer different data processing, storage and meta-data analysis tools and features. Furthermore, these gateways need to allow for migration and clustering while maintaining device communication and application persistence within the cluster and the Fog. Current approaches fail to provide an application environment to decouple deployment and messaging which is partly due to a lack of M2M protocol abstraction. They also fail to provide a virtualisation layer for applications to allow complex application deployment using the resources of a set of gateways rather than the limited resources from a single gateway. This becomes a particularly big issue for use cases where a highly interconnected and constantly reconfiguring environment is in place such as in the case of Smart Office and Home scenarios as well as Industrial Monitoring and Control applications that involve task and project based reconfiguring of the system.

2.5 Application and System Model

Future IoT and Fog computing systems will be deployed in an environment that is highly heterogeneous both in its geographical distribution, density and in the characteristics of both the applications and their host nodes. The hosts vary from highly heterogeneous edge networks with reduced resources to Cloud nodes that can be considered as having endless resources. The introduction of these concepts promises to reduce delays, increase reliability and improve security as well as provide better resource utilisation. In order to achieve this however, services or applications need to be deployed as optimally as possible. Before this can be achieved however, there is a need for the requirements of such systems to be analysed and models to be put up so the health of each individual service or application as well as the systems' health can be evaluated, monitored and estimated. An attempt of finding the platforms' general characteristics has been carried out in (Cruz et al. 2018) where they do not provide a mathematical model that can be optimised but rather a collection of all the characteristics that can be considered in Gateway based Systems. The proposal in (Jim Zw Li et al. 2011) provides a model that can account for several parameters in a multi-goal model. A combination of models that can account for goal heterogeneity as well as multiple parameters is crucial for these systems.

The types of models proposed by researchers are linked to the optimisation method attempted and the use-case context. Cloud and Edge applications require different models

while Fog use-cases require both to be considered. Some research focuses on providing very accurate but narrow models while others seek to encompass a wide range of parameters but using simple methods. The issue of latency is a key factor in Fog computing. The authors in (Bauer, May, and Jain 2014) propose a real-time gateway approach for Industry 4.0 where they seek to model the latencies in the Ethernet system through the slave and master components. The framework in (N. Wang et al. 2017) is built on the idea of optimising resource use while reducing latencies in the system by a reported minimum of 20%. This is done by deploying services to the edge nodes that are closest to the users. The research carried out in (Intharawijitr, Iida, and Koga 2016) focuses on modelling 5G network based systems to reduce communication latencies through the Fog model. Their approach looks at varying latencies and types of connections as well.

Other approaches may include reliability, energy, QoS improvement or SLA compliance and even pricing calculation and reduction. The framework proposed in (Osanaiye et al. 2017) looks at providing a live-migration backbone for Fog environments based on VM, and seeks to model the Reliability and Availability of these. The computational offload technique in (X. Chen et al. 2016) looks at offloading code and workloads from the Cloud to the edge in such a way that they model latencies, computational energy and overhead. The research of (Zeng, Gu, and Yao 2018) proposes a more advanced energy estimation model that focuses on computational and networking costs, as well as seeking to minimise these. The approach in (Congjie Wang et al. 2017) focuses on multi-Cloud deployment through modelling QoS compliance. They also propose an optimisation algorithm for both global and load balancing deployment styles. The model in (Aazam and Huh 2015) looks at methods of estimating costs related to deployment in Fog data centres based on resource estimation. It attempts to calculate resource scarcity. This is then evaluated using the CloudSim toolkit.

These models can differ in their application environment as well. Some are designed to work in a way where applications are considered to be deployed one after another or in some cases little or no information is known about these linking components. Load balancing approaches usually look at applications and their components as a subset of tasks or workloads totally independent of each other that need to be allocated within a time-frame satisfying some criteria. The authors in (Minh et al. 2017) look at decomposing applications into their tasks to better deploy them on both Fog and Cloud nodes. Their model focuses on resource utilisation modelling. The proposal in (Skarlat, Kevin, and Schulte 2018) proposes a workload view of services where they are deployed on the Fog and Cloud system with full knowledge of their demands. They look at deployments where response time constraints are not violated and the total execution time is reduced. These can be categorised by a simple and easy to implement light-weight model that is being used in conjunction with greedy

or first-fit algorithms. The work in (Taneja and Davy 2017) looks at resource utilisation and latency modelling and global deployment. Their model is designed to consider all the connections and the algorithms are designed on top to find the minimum. The drawback of such approaches is that the resulting model that needs to be optimised creates an NP-Hard problem in most cases. The proposal in (Intharawijitr, Iida, and Koga 2016) attempts to minimise a connection graph of components deployed in a system which follows the global deployment direction.

The presented models all vary in how complete they are, in how accurately they describe the systems they attempt to model, and how complex these systems are in real life. Cloud based systems typically require simple models as they are considered mostly homogeneous. The Fog or edge based counterparts are usually more complex and consider different variables as they are more heterogeneous in nature which makes creating such models more difficult but also gives more opportunity for improvement. Optimising these systems using some of these models results in an NP-Hard problem, which is a main concern for deployments as their scalability is affected by a non-polynomial increase with size. A trade-off of complete models and complex models needs to be reached where the models are complete enough to be accurate but do not fall into the NP-Hard problem category.

2.6 Deployment in the Fog

Several CPS Clouds have been proposed to enhance the functionalities and alleviate some barriers in CPS development (Chaâri et al. 2016). However, latency is one of the inherent challenges in Cloud computing, and this issue limits its use in time-sensitive applications. CPSs often work in a dynamic environment having a mixture of urgent, unexpected, and periodic events with hard and soft real-time constraints. This means the Cloud approach may be inadequate for CPSs (García-Valls, Cucinotta, and C. Lu 2014).

The core concept behind Fog Computing optimisation is to improve the efficiency of resource utilisation throughout the system. Load and Delay optimisation as suggested in (Dhinesh Babu and Venkata Krishna 2013) is imperative for time-sensitive CPS applications and has become an important field of research for managing resources in Fog environments. The use of OSGI and similar resource sharing platforms for application deployment (compared to Container and VM based solutions) offers new possibilities but also creates challenges due to increased interdependence.

When considering Application deployment in either Fog or Cloud environments there can be two approaches to solving the problem. Load Balancing techniques focus on online or on-the-fly optimisation where requests or changes need to be satisfied as they arrive

and reaching a solution in an allocated time set and having an acceptable solution is more important than having an optimal or improved solution (Sivieri, Mottola, and Cugola 2016). The second method focuses on Global optimisation where a known set of initial requirements and set-ups are given as well as workloads and system configurations. A solution is then required to deploy all of these on an system. These methods can be used interchangeably or can build on top of each other .

2.6.1 Load Balancing

Various resource management strategies and algorithms have been studied for decades in a variety of scenarios and it is a well understood area in general distributed systems and Cloud computing. However recent years have witnessed that researchers are moving their focus towards load balancing optimisation for Cloud-Fog systems. The key question in load balancing optimisation is how to allocate jobs on various machines so that each receives its fair share of resources to make progress while providing good performance (G. Lee, Chun, and Katz 2011).

The studies conducted by (Lucas-Simarro et al. 2013) and (Zhan et al. 2015) provided comprehensive survey of various techniques for load balancing optimisation for Cloud computing. The load balancing optimisation mechanisms used in managing resources in the Cloud computing context can be broadly be divided into virtual machine (VM) and task based allocation optimisation. These focus on optimising parameters such as QoS, Load, Costs and Energy. (Hu et al. 2010) presents an algorithm for load balancing optimisation of VM resources by using a genetic algorithm. Their proposed algorithm attempts to reduce migration cost of VMs using historical data and current state of the system. (Zhao et al. 2013) presents the design and implementation of an algorithm that employs the Pareto dominance theory and simulated annealing to achieve a long-term efficient power saving and load balancing optimisation. (Dhinesh Babu and Venkata Krishna 2013) propose an algorithm to achieve load balancing across VMs in order to maximise and balance the priorities of tasks so that the amount of waiting time of the tasks is minimal (Trappey et al. 2016). A task scheduling algorithm for load balancing optimisation based on QoS, proposed in (Wu et al. 2013), computes the priority of tasks based on some specific attributes and evaluates the completion time of each task and then schedules each task onto a resource, which can complete the task according to the task priority. In (Ramezani, J. Lu, and Hussain 2014) the authors propose a new load balancing optimisation method that uses Particle Swarm optimisation to balance system load by transferring tasks from an over-loaded VM to less loaded one.

Through application of various algorithms, these approaches revolve around optimal task distribution on various VMs or VM migration to achieve effective load balancing optimisation. The underlying characteristic of these algorithms requires their customisation to be applicable in Fog computing settings.

The Fog computing model extends the Cloud load balancing problem from Cloud resources to include edge device resources. The Cloud load balancing optimisation methods cannot be applied in Fog computing due to the fundamental difference between these computing models. The Fog computing dynamic load balancing optimisation mechanism provided in (Ningning et al. 2016), through graph partitioning and clustering, assigns tasks to VMs according to the resource requirements of the tasks. The authors propose this method as they conlude that, existing Cloud optimisation methods for load balancing have shortcomings in terms of system hierarchy and load forecasting, which cannot be applied to the dynamic and P2P architecture of Fog computing. The proposed research, on the other hand, focuses on application migration.

In (Deng et al. 2016) a framework has been proposed to investigate the trade-off between power consumption and delay in the Cloud-Fog environment. The research in (S. Verma et al. 2016) proposed a load balancing optimisation algorithm, which uses a data replication technique for maintaining data in Fog networks with an attempt to reduce overall dependency on big data centres. Application latency in Fog computing has been addressed by VM migration (L F Bittencourt et al. 2015). In (He et al. 2016) an architecture is proposed to integrate Fog computing and SDN (Software Defined Network) to IoV (Internet of Vehicle) to improve the latency of sensitive services. This approach is highly domain dependent and uses particle swarm optimisation to decrease service latency.

2.6.2 Global Optimisation Techniques

Global optimisation, as opposed to Load Balancing and similar solutions in the context of application and service deployment, can be defined as an attempt to allocate all the components as if they appeared at once. Load balancing techniques can handle similar situations but their solution, as well as their response style, might be far from optimal.

Cloud computing approaches as in (Congjie Wang et al. 2017) can be described as data centre based, with homogeneous resources and low latencies. The optimisation techniques deployed on these systems reflect this. They employ methods to decrease costs or improve SLA and QoS adherence. The paper in (Jim Zw Li et al. 2011) looks at multi-parameter optimisation based on SLA, costs, licensing, etc. to deploy a set of applications in a Cloud environment using a Greedy Algorithm. The approach in (J. Li et al. 2009) considers an Auxiliary Network Flow model based incremental method to decrease costs in a Cloud

deployment. The framework in (Beran, Vinek, and Schikuta 2011) focuses on optimising profits across a large set of workloads and parameters based on performance models. The deployment strategies used in (Congjie Wang et al. 2017) propose a QoS aware content allocation based on a prediction model and a first fit method.

The Fog Computing approaches connect the Cloud with the Edge of the network which means that the new environment will have an increased heterogeneity as well as varying latencies. In this scenario the edge nodes are more resource constrained and different nodes have different requirements. The solution in (Taneja and Davy 2017) proposes a custom model based on use-case scenarios that maps application modules to network or Fog and Cloud nodes. This method uses a first-fit method to place apps on the system. Deployment in large-scale heterogeneous smart environments can be seen as Fog deployments as suggested in (Cicirelli et al. 2017) where they investigate a container and VM based deployment framework. The authors of (Minh et al. 2017) propose a Fog based Service placement method that differentiates between Cloud and Fog services and attempts to maximise the number of services deployed on the Fog. The research in (N. Wang et al. 2017) addresses the geographical distribution of data centres or Cloud servers to manage a set of edge nodes through their ENORM framework using a simple First-Fit method for deployment.

The review paper in (Cruz et al. 2018) evaluates existing platforms and IoT Systems to identify key characteristics, components and modules. These can be used to define and apply a Fog model for deployment that can satisfy an increased number of use cases. Optimisation methods can then be tailored to these characteristics.

Container based optimisation has emerged since the rise of Docker (Merkel 2014) as a lightweight alternative to VM's. The authors of (Hoque et al. 2017) look at optimising container based deployments in IoT based Fog and Cloud environments. Cluster based deployments orchestrated by Kubernets (Brewer 2015) is then proposed. The PaaS based approach in (Aggarwal and Aron 2017) focuses on providing an architecture to support container deployment in IoT environments to better utilise resources. The research in (S. Kim, C. Kim, and Jongwon Kim 2017) proposes a high level IoT polyglot framework that can encompass and orchestrate a varying number of Cloud and edge clusters.

These methods usually employ greedy or first-fit methods to deploy applications, due to the large-scale of typical Fog systems. While these provide an attractive time-complexity, their solutions are sub-optimal. More advanced methods need to be considered to fully utilise the capabilities of the Fog. The methods proposed in (Zeng, Gu, and Yao 2018) are designed to solve the NP-hard problem of service composition in heterogeneous environments through an algorithm that attempts to reduce the complexity of the optimisation problem. The research in (Duro, Purshouse, and Fleming 2018) investigates methods of using partitioning or

clustering to divide complex system into a set of smaller subsystems to reduce the complexity and time required to find a minimum. The authors of (Intharawijitr, Iida, and Koga 2016) focus on evaluating Fog computing as a paradigm to find size and scalability issues as well as inflexion points where the use of clustering or fragmentation is warranted.

2.7 Network Analysis and Clustering

Optimisation in IoT systems is made difficult by two main factors. The first is with respect to the complexity of these systems where migrating an application, service or resource leads to the alteration of the connection topology as well as the locally available resources the effect of which is difficult to model or estimate. The second hindrance in developing an optimisation solution for Fog computing is the lack of real-life information on data-sets, use-cases, application sizes, processing requirements, message rates and their impact on the deployed nodes. Most available use-cases such as in (Scheuermann, Verclas, and Bruegge 2015) show a high-level view of agile manufacturing systems which cannot be used for optimisation purposes. The state of the art solutions for this as in (Oueis, Strinati, and Barbarossa 2015; Deng et al. 2016; Zeng, Gu, Guo, et al. 2016; Vogler, J. Schleicher, et al. 2016) are the proposal of example applications and use cases on top of which they build their optimisation methods. The drawback of these approaches is that there is no guarantee that the proposed system parameters or use-cases resemble real-life solutions, reducing the utility of the proposed models and algorithms.

The solutions look at different aspects of optimisation. The solution in (Oueis, Strinati, and Barbarossa 2015) is a clustering based method based on a simple delay model between components, while in (Vogler, J. Schleicher, et al. 2016) a simple topology reduction is attempted. The proposals in (Deng et al. 2016; Zeng, Gu, Guo, et al. 2016) show a more elaborate application and delay model considering processing delay as changing through the deployment locality as well as considering several different connection delay types. Although these models are extensive, the constants, rates and values that change through migrating are assumed instead of measured or deduced. This may cause certain optimisation approaches to seem more advantageous than others as well as leading to inaccurate models.

When considering highly connected complex systems, the common approach is the use of graphs to model the connections between entities. This has been done to model WWW connection as in (Kleinberg et al. 1999) as well as to optimise Wireless Sensor Networks (WSN) as in (Savazzi, Rampa, and Spagnolini 2014) for connection reliability, zero single node failures and other parameters. Increasingly there are attempts at using these methods on Fog Systems as in (Jingtao et al. 2015) where a tree based system is used or in (Ningning et al.

2016) where graph re-partitioning methods are proposed. These proposals have the same drawbacks of lacking real deployment data on which to test their algorithms on real-world systems where the clustering factor, connectivity and distribution of nodes might vary greatly. Finally, these solutions do not consider the existence of a physical and virtual connection sets, where the physical one consists of where application, devices and resources are deployed or orchestrated, while the virtual one consists of interactions between components. The mapping of the virtual graph to the physical one is the core of the Fog Computing placement problem.

2.8 Summary

Advancements in Sensor and Actuator technologies and the standardisation of protocols have led to the Emergence of the Internet of Things. This paradigm shift proposes the interconnection of billions of devices through a varying set of methods. The resulting environment is a highly interconnected heterogeneous system that promises to solve some of the main issues of implementing Industry 4.0 requirements. The requirements of this proposal envisage factories with interconnected devices and systems that would result in increased productivity, higher range of feasible products and increased safety and fault response. To achieve these goals, the IoT systems need to have reliability and latency parameters at the edge of the network that are usually reserved for the Cloud.

The paradigm shifts that came with the introduction of Fog Computing by Cisco aims to solve some of the latency and reliability issues that some of the core IoT frameworks and approaches have by looking at the system not as separate Edge and Cloud components but as one whole Fog. This approach envisions extending the virtualisation that can be found in the Cloud to the edge of the network utilising the resources available at the edge for lower latencies and data locality. These concepts can be used in Industry by employing local or on-the-fly processing of sensor data, which can reduce latencies and decrease the reliability and security issues that come with sending sensitive data up to the Cloud.

To achieve the virtualisation level suggested by these new paradigms on the edge devices, a shift from traditional VM based deployments needs to take place. This is required by the nature of the edge nodes that have fewer resources than traditional Cloud nodes and might be at higher demand due to their close proximity to end devices. Container and shared environment based solutions are proposed as a way of keeping some of the separation present in VM based deployments while providing a more light-weight deployment. These gateways also need to consider advanced networking discovery and management solutions that can adapt to a dynamically changing environment. Furthermore, these gateways need to support

multiple application types of varying languages as well as have connection capabilities to several types of peripheral devices.

Deploying applications on these gateways is as big a challenge, as is creating the gateways themselves. These applications can be as heterogeneous as the gateways themselves with varying requirements and characteristics. They may require large processing capabilities, or have these vary over time or with the number of devices. These could require high reliability or low latency. To accommodate these, the first thing that needs to be done is to model how the gateways and the applications behave so the problem domain is known and so is the 'Hardness' of the problem. The optimisation methods attempted in the reviewed papers as well as the models vary in complexity and applicability where some only attempt to do a First-Fit or Greedy deployment with emphasis on fast solutions, while others attempt global deployments with GA, PSO and similar approaches.

If the requirements of Industry 4.0 are to be met through IoT and Fog Computing, a lot of research still needs to be done so a standard framework or set of platforms can be used. Available and accurate models that can be used together with a varying number of load balancers and global optimisation tools are needed to provide the users and developers with ways of measuring deployed systems and also deciding on what actions to take or on which use-cases they can be used.

Existing solutions fail to recognise that the system models need to be based on measurable parameters and contain as complete a characterisation of the system as possible. With the proposed optimisation methods, these need to be tested and analysed on use-case driver scenarios and their limitations or preferred use-cases need to be determined. Finally, tailored methods need to be proposed that can answer the large-scale and interdependency requirements of future Fog systems.

Chapter 3

Fog of Things Platform

3.1 General View and Platform Requirements

The proposed Platform as a Service generic gateway architecture attempts to answer the requirements of an ever-evolving IoT environment while improving on existing proposals especially on the topic of migration, clustering, abstraction and routing of device messages to the appropriate regions. The use of the resources available on the Gateway has been expanded from those suggested in (Aazam and Huh 2014) with the introduction of context information such as region, network information and location information. The review showed a need for a generic architecture that can encapsulate a wide variety of containers and drivers from different providers and languages.

This architecture is designed to fulfil the requirements stated in the below subsections.

3.1.1 Protocol Agnostic Device Messaging

The messaging between devices and the application environment through the drivers is designed to allow for messages to be transmitted regardless of the devices' protocols or technologies. This allows applications to be oblivious to the underlying protocols or technologies with which they want to communicate. Furthermore, due to the routing of messages, applications can communicate with the devices from the Cloud, or with the ones that are registered to other gateways on the local cluster.

3.1.2 Regional Connections and Messaging

When gateways are deployed onto a WAN network they can form a local region which should allow information and messages to be shared between peers. This allows for faster message

passing among local devices and with this connection clustering and high availability are also possible.

3.1.3 Multi-Cloud Tenancy

The gateway should enable multiple Cloud connections to be established in order for application and management information to be sent and received from these tenants.

3.1.4 Modular Application Deployment

The application container should allow multiple applications to be deployed on the same gateway and communicate with each other so that complex applications can be deployed across simple components.

3.1.5 Application Migration, Clustering and Testing Functionality

Due to the nature of the gateway, it needs to meet QoS requirements associated with the applications or Cloud that it interacts with. These applications need to be tested and migrated seamlessly while maintaining inter-application and device communication in a secure environment.

3.2 Generic Gateway Architecture

This research defines the Fog of Things as a Fog Computing platform that treats things as resources of the edge device and allows for a unified view and messaging with these devices. Fig. 3.1 shows the overview of the platform and the connections between components. The proposed gateway architecture is built around a new asynchronous messaging based model that allows the abstraction of different drivers and components by allowing messages to be routed to their destinations dynamically, based on a new header oriented routing model.

The proposed architecture offers a novel gateway design by increasing the horizontal integration of the gateways by allowing applications to send and receive information to and from a number of Cloud providers using the configurable brokers. It also offers a wider range of client connection possibilities by providing WAN client connectivity through the configured regional connections. Another novelty presented by the gateway is the protocol agnostic container environment that allows applications to communicate with Cloud providers, regional clients, peer applications, devices and requests resources through a unified medium without considering the underlying protocol for device, region or Cloud

communication. This is achieved through a set of brokers and drivers that translate and route these requests into messages understood by the respective sinks. The final novelty of the gateway is the possibility to configure WAN clusters of peer devices and migrate applications without the need of reprogramming them between the peers and available Cloud containers.

This architecture in Fig. 3.1 and the associated components described in the following paragraphs can satisfy the requirements presented in the section above.

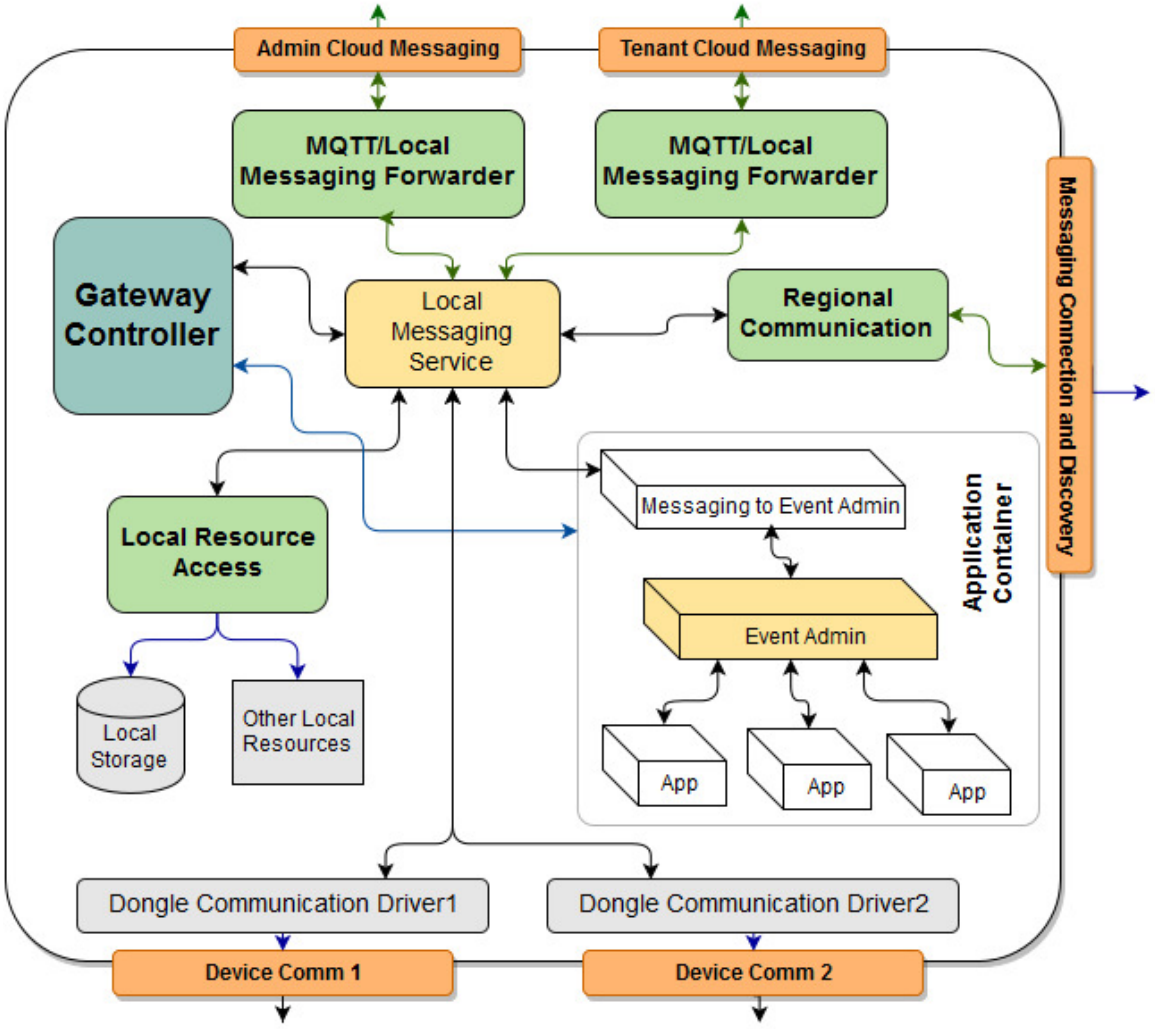

Fig. 3.1 Architecture of the Gateway

The gateway controller analyses and deploys applications, as well as sending usage, load, capacity, connected device and region information to Cloud and region clients. The gateway manages the non-admin tenant connections and the device drivers, and controls the regional authentication and registry. Finally, the gateway is capable of searching for available gateways in its WAN network. It can either enrol them to a region or create one and become

its coordinator, if no peers are found. The controller manages the information about the capabilities of the gateway, its resources, the connected drivers and the available regional devices.

The Application Container is controlled and monitored by the Gateway Controller. Rather than having applications connecting to the Messaging Service directly, the Application container translates messages and events into its internal equivalents that can be understood by the deployed applications. This allows more applications to listen to the same broadcasted message, communicate between each other, and send information to the outside components asynchronously. Furthermore, this allows policies to be put on the devices like an internal firewall that would allow apps to send and receive messages only from authorised or authentic sources.

M2M communication is fulfilled by the device communication components that are directly linked to the transceiver hardware and are also tasked with registering, authenticating and monitoring the devices. The received device messages are interpreted and sent to the corresponding sink through the messaging service, while messages sent from applications are encoded into the desired format and sent to the devices.

Cloud communication takes place through dedicated brokers that take messages directed at them, parse the headers and payload to the desired format and send them through the broker's medium, doing the reverse for received messages. This allows for different protocols to be used by tenant Clouds to access applications and the gateway controller. Storage and metadata information like location, regional clients, network information and other gateway details are considered local resources to the applications. Applications and devices are allowed to save data into databases, request location data and send the data to the application layer or the Cloud.

The regional communication refers to two distinct communication methods. The first looks at gateways that can be discovered through a local network and that can be linked through the federation of the messaging service. The second method proposes the creation of regional access points to applications which can receive messages from a more varying range of local clients.

The details of the proposed components are described in the following subsection. Each section looks at a major component of the framework and describes its functionality and proposed mechanism.

3.2.1 Local Messaging Service

The local messaging service is responsible for routing messages to the appropriate queue based on their headers and routing information. It is designed to support asynchronous

messaging between components. Furthermore, new drivers and different configurations can be added to the gateway without modifying any applications or other components. In order to accomplish this, the messaging service is designed with a complex array of exchanges, which can be seen in Fig. 3.2.

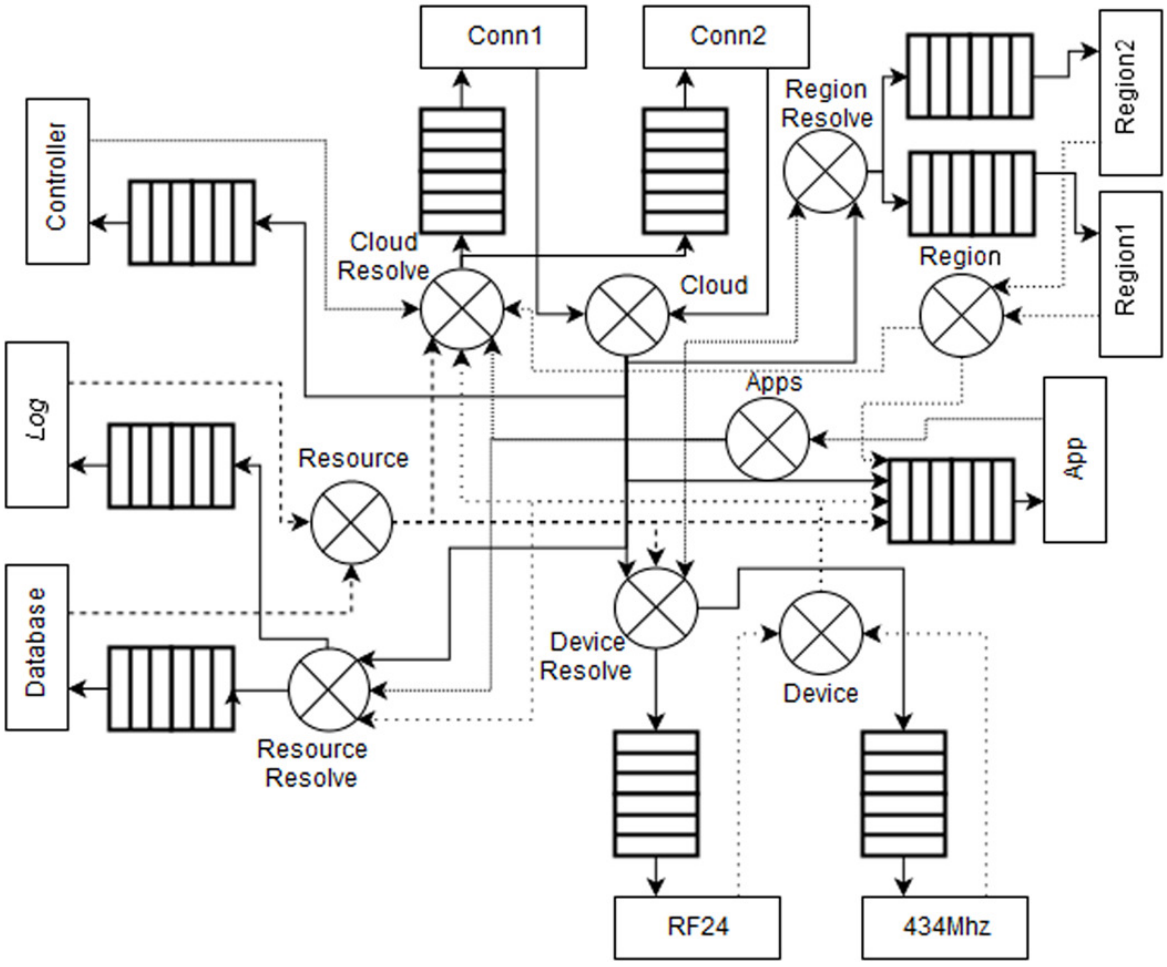

Fig. 3.2 Messaging Exchanges and routing

The routing is designed in such a way that components can send messages in a generic format and the exchanges can route these messages based on the routing table on the gateway. The exchanges that route messages to other components hold the group name of components (resources, devices, region, Cloud, apps) and are designed to route the collected messages to the corresponding resolver components.

The message passing is designed for scaling, in order to support the addition of new components seamlessly and removal of old ones. Resolver exchanges allow messages to be routed to their specific queues based on header information and are the main configurable components to support the routing table in the messaging service.

Components are designed to communicate with other components by publishing messages to their specific exchange and retrieving messages from the queue in a unified way, without knowledge of the number or type of destinations of the message. This takes away the burden of re-configuring the components when modifications on sources or destinations take place.

The control component is a special one, as it does not communicate with any other components through the messaging system but configures them on deployment, with the exception of the Cloud connections which it uses to send and receive information and control parameters. The region component is connected to the container and Cloud component, which is done in order to be able to route messages to applications which are deployed locally and to those which are deployed to the Cloud.

Each message on the system follows the same basic link from their source to their destination. At first they get routed to an exchange based on their loose connection. The main exchanges are: Cloud; Region; Apps; Resource. These regions that route the messages based on their headers, information and specific source to a resource resolvers exchange as: Cloud Resolver; Region Resolver; Apps Resolver; and Resource Resolver. These then route the message to the desired messaging queue where the right sink will be able to read it. As example, a message from a cloud service to an application will come through the Tenant Cloud Messaging Broker from Fig. 3.1 that sends the message to the Cloud exchange, which based on the headers, routes the messages to the Apps Resolver Exchange. This exchange then routes the message to the Apps Queue that is then read by the Messaging to Event Admin Broker that translates the message to an internal event in the Application Container. After this, application designated as receivers of the message can catch the event.

3.2.2 Cloud Controller and Local Resources

The Cloud controller is responsible for configuring and deploying all the communication drivers with the Cloud or the devices as well as managing the regional connections and authentication while relaying status information to the Cloud.

The gateway sends status information to the specific Cloud component by responding to requests that were made through the Cloud connections. The first and main Cloud connection has the most control over the system, as it is able to add and delete other connections, remove and modify apps deployed by other tenants as well as set up the region communication and the device drivers. The other tenants are limited to offering and requesting authentication information for devices or regional agents as well as deploying and configuring their own applications and devices.

Local resources are controlled by the gateway controller and receive requests, data and commands through their respective drivers connecting them to the local messaging service

and through this the applications. These local resources may include context information such as region parameters, location information, and storage. The storage component is a special one, because in contrast with other resources, it can contain meta-data to support data requests. The resource can be configured for high-availability throughout the gateways as well as through Cloud backups by replicating its functionalities which are abstracted away from the applications. A distributed database is proposed for the use of Cloud storage and redundancy is proposed for highly distributed unreliable systems. Local versions will run on the gateways providing local instances of data, smart migration and backup.

3.2.3 M2M Communication and Registration

Each gateway is equipped with its own set of communication mediums to transmit and receive messages from sensors and actuators. To account for differences in communication protocols and communication mediums the gateways have a driver for each medium that acts like a broker between the devices and the messaging system. These brokers are used to authenticate devices and add them to the locally available list for security and encryption. Furthermore, they interpret the received messages and assign the proper routing and header details to assure that they reach the required destinations. These drivers can have a more diverse range of tasks based on the requirements of the protocols and mediums such as packet forwarding, routing table creation and other WSN gateway tasks.

The registered device information is stored in the driver specific database and is used by the gateway controller to determine which application to deploy and for routing purposes. Furthermore, the device information saved in the database, that uniquely identifies the connected physical devices and their states, is used by the driver to monitor, authenticate and correctly route messages to their destinations.

Due to the wide range of protocols and transmission mediums, the messaging and routing system needs to be configured in such a way to allow different drivers to send and receive messages in a unified way. A slightly altered version of a JSON based markup language presented in (Jennings, Arkko, and Shelby 2012) that has been used to link advanced IoT structures in (Lampesberger 2016), called Sensor Markup Language or SenML is proposed. This would require all connected devices to register, send and receive information based on this language. The device registration information needs to contain information about the device's type, its version, and the sensors that it is equipped with. Any other communication specifics that are not relevant to applications or monitoring of devices are abstracted. The SenML based device message transmission is used for driver to driver, driver to app and app to driver communication only and is used as a common medium between protocols that can describe messages that need to be sent. The actual messages sent to the devices may

vary depending on the control protocol. This would allow older devices to use their existing handshakes and means of message transmission to be connected to the system.

3.2.4 Application Container

The client bundle in the container can be configured to read messages from a messaging service queue and to create events based on these messages. The applications create events, the broker reads the messages generated from these events and sends them on to the local messaging service as shown in Fig. 3.2 The headers of the messages are designed to allow applications to send messages to different locations, but also to act as a filter between the application container and the gateway resources only allowing applications to send messages to their preconfigured resources.

Communication between applications can be carried out in two distinct ways. The messaging system is more suited for communication between applications that are not closely linked to each other and can be interchanged. Communication through the internal services or other structures provided by the container is more suited for use within the same application set to create larger application from individual bundles following the Microservices architecture. The only constraint is that applications that communicate with each other through container specific structures need to be migrated together. Those which communicate through the messaging service can be kept in different locations and migrated separately.

In order to enable applications to respond to new devices being added to the system as well as to be able to listen to individual devices and have messages transmitted to these applications from the Cloud, applications need to be able to reconfigure their application name and the name of the devices they are listening to. This is achieved through assigning a configuration file to each application that contains all the relevant information. The gateway controller adds information regarding the devices the application is configured to communicate with as well as the applications' name, the regional communication channel, the Cloud connections and other configuration parameters. When the configuration is updated, all applications are refreshed to start with the new set of data. Applications can then be deployed into multiple environments with multiple use cases as well as facilitating their testing and migration.

The construction of the application container allows for application migration within the local cluster and to the Cloud. One of the main differences between the application container on the host or other local gateways and the Cloud based/virtual gateways is the complexity of the messaging service. The gateways residing on the Cloud only receive and send information from one source, having the modified brokers in the application container mimic the gateway

sources of the messages based on the message headers. This difference allows the deployed applications to be location agnostic, receiving messages in the same format. Finally, using this method, the creation of virtual gateways is possible as well. These virtual containers have dummy applications that mimic the behaviour of real devices by posting and consuming events on their behalf to allow for a more realistic testing environment as well as for scaling experiments.

3.2.5 Regional Communications and Clustering

Regional communication refers to gateways that are on the same network or can reach each other through local network scans or any other methods that send and receive application messages for clustering, high-availability or other inter-application communications. There are two ways to connect and access applications from the local network. The first one, with more constrained connection is realised through the federation of the messaging service, so gateways can connect to each other seamlessly. The second one is a more loosely coupled connection that would allow messages to be sent from different clients through the regional drivers that convert the messages to application messages inside the messaging service. The federation configuration of the messaging service offers better security, message latency and ease of use due to the fact that it extends the messaging service from one device to another by having the exchanges mirror on all nodes and having some of the queues unique to their specific gateways. Applications can be deployed on a single node and communicate with other devices and Cloud tenants. The federation messaging approach also enables devices to configure clustering and high-availability as resources which may lead to better QoS parameters.

The more loosely coupled connection through the regional drivers would permit gateways to be of different types and configurations with even outside applications connecting to these endpoints. The configuration of these endpoints would be fulfilled by the Cloud controller that creates a queue for each application that has regional communication set up in the configuration files and modifies the driver to make these available through external requests. These requests are treated as RPC calls and each request has a unique transaction id. This solution offers extra functionality and reduces costs by adding an alternative of accessing applications through the local network rather than through the Cloud connections.

3.2.6 Cloud Connection and Management

Cloud connections enable the gateway to send and receive application data, sensor and actuator data, as well as to migrate applications through message passing and the deployment

of applications. Each connection to the administration or tenant Clouds is managed by a designated broker through the protocol preferred by the Cloud. The first connection is to the main Cloud, which is preconfigured in the gateway. The other connections can be started through commands received on the first one using the gateway controller.

When applications are migrated to the Cloud, the respective connection is used to allow messages that would normally be transmitted to the local container to be transmitted to the Cloud where they are routed to the Cloud container. The brokers in the container transform them into messages with the headers and payload corresponding to those received on the physical gateways container. Applications can be migrated from the physical gateway to a virtual gateway in the Cloud while retaining all inter-application communication and local messaging without reconfiguring or redeploying the applications.

In order to allow inter-application communication to occur a forwarder is required in the container that receives messages designated to the application and sends them to the Cloud communication component as well as accepting responses and creating events as if the application was never migrated. This functionality can be extended to replicate local services on the container.

3.2.7 Migration and Message Routing on the Platform

The message routing on the gateway is the backbone of the protocol agnostic messaging for drivers as well as the mechanism used for the migration of applications between gateways. Migrated applications need to maintain their full functionality, being able to access data in storage and all connected devices while being able to communicate with peers in the region, Cloud and other services on the gateway.

The migration process requires all messages going from and to the application to be routed to the new host gateway. This gives the app the illusion of still being on the same gateway, without needing to reconfigure or rewrite its code.

The messaging service uses federated connections to forward messages between peers. The connections between gateways by default are in a star topology that allows each gateway to access each other directly, reducing latencies and hops but increasing overhead on larger systems. Other topologies can be designed per application environments.

An example of how this routing is done can be seen in Fig. 3.3 where Application 2 is moved from the Cloud VM to the gateway, this changes the run-time characteristics of the Application, the new environment having different latencies, different load and processing capabilities.

An application is migrated by deleting it or stopping it on the host gateway, re-configuring the existing routes of the application to be sent to the new host, adding the routes on the new

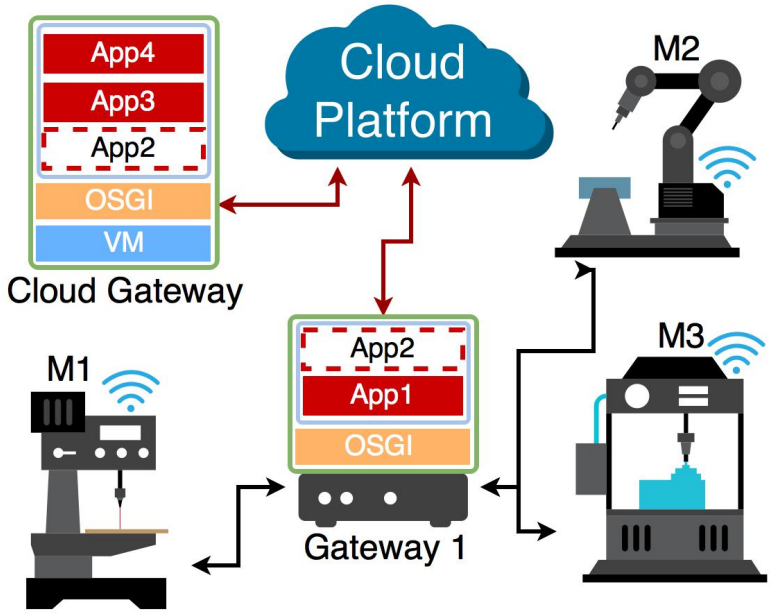

Fig. 3.3 Application Migration from Gateway 1 to the Cloud Gateway

host to the application container and then finally deploying and starting the application in the new host's container. This is done through the configuration file of the application.

3.2.8 Application and Gateway Monitoring

The monitoring on the gateway is done by two components. The first is inside the application container and monitors all application messages, even inter-app messages as well as the total CPU usage of the applications threads. This component sends a message to the gateway monitor which looks at a wider range of parameters but takes a more general look at messages. The second component creates a summary file which is saved to the database periodically.

The monitor in the application container is able to retrieve information on messages sent to and from each application to any drivers, Cloud connections, regions and resources. Furthermore, it reads information on the CPU usage of every application. The gateway monitor has a more general view of the messaging as it shows all messages routed from all components without information on individual users/applications. This monitor can also give information on individual application storage use, gateway load, RAM use and CPU usage on the system. The load is adjusted to the processor count of the platform.

3.3 Architecture Implementation

The existing architecture is implemented based on the general descriptions and technical requirements presented in Section 3.1. The implementation demonstrates the feasibility of the proposed generic gateway architecture as well as the use of the OSGI container as a gateway application container. The communication technologies and message passing system is chosen based on the literature review of the existing platform and current technology trends done in the Sections 2.2 and 2.4. The underlying messaging architecture is AMQP within the RabbitMQ server. The proposed communication mechanism with the Cloud is MQTT which has received support from an increasing number of Cloud providers. For the regional communication, either REST or STOMP based drivers are proposed while the clustering of gateways is supported through the federation functionality of the RabbitMQ messaging server. Each M2M communication protocol and device has its own functionalities, advantages and drawbacks. The device drivers' subsection shows the basic backbone to the drivers that were used. For the application container, the OSGI based Karaf is the most compliant with the proposed generic architecture.

3.3.1 Device Drivers

The approach to creating the drivers has been tested for four different communication mediums, 434Mhz, RF24, Bluetooth and Xbee. These four mediums differ in their level of abstraction of the OSI layers as well as in their added functionalities. The first protocol only implements the physical level requiring the driver to configure the rest. The RF24 based protocol has the added functionality of discovery and being able to listen to specific channels, but what this lacks is the ability to listen and communicate on multiple channels effectively. The Bluetooth based RFCOM communication protocol allows for a wider range of features and sending messages through sockets to certain devices. Xbee is a similar protocol having multi-hopping and networking functionalities as well. The drivers for these protocols work in the same way for applications, with none of the differences being visible at the application level.

All applications to be deployed in the proposed architecture at least include a few common functionalities and these are: the registration of devices; the monitoring of devices; and the sending and receiving messages from devices based on their ID's. The whole registration procedure is shown in Fig. 3.4 for the case where the device has been previously registered or when it is a newly registering device.

After the registration, devices only send a shortened version of the sensor data, only containing the sensor name, the value and the device id. The received message is parsed to

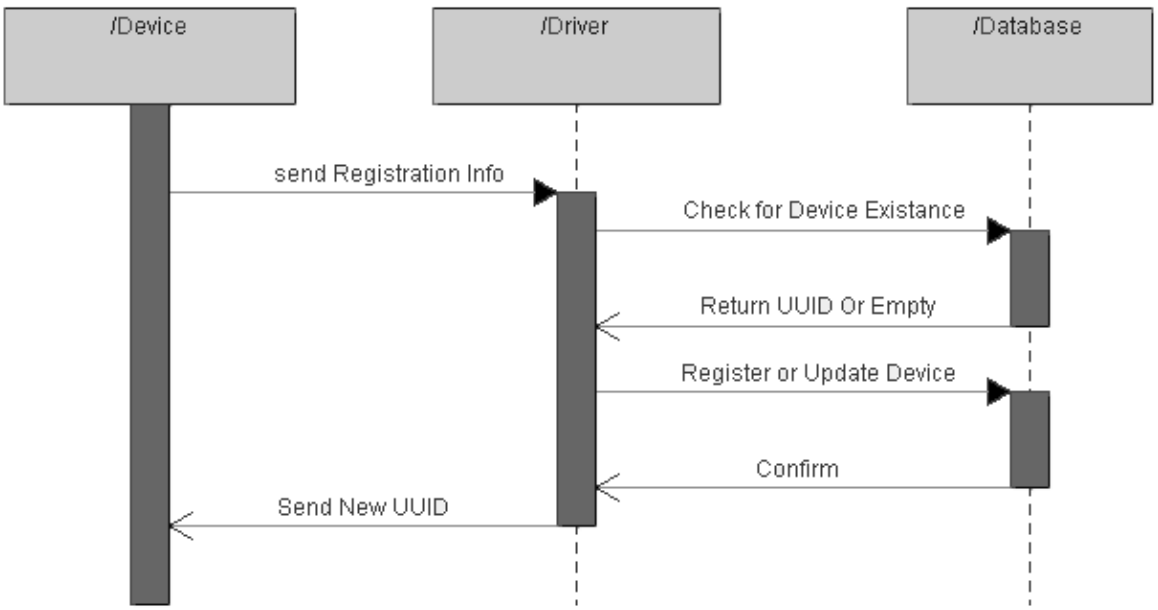

Fig. 3.4 Registration sequence diagram

make sure that it is consistent to the JSON format and key information like the device id is extracted and then the appropriate header information is created and the payload is sent to the messaging service. The structure of this message can be seen in the example shown in Table 3.1.

Table 3.1 Message from Driver

Content Name	**Data/Property**	
	Property Name	**Property Value**
Header	device	OWaDMY9V
	dev_type	ardUnoTemp
	dev_count	0
	comm	Gateway-RF24
	datetime	2017-05-09 12:02:36
Payload	[{"v": "26.00", "n": "temp"}, {"v": "34.00", "n": "hum"},{"v": "8.95", "n": "dew"}]	

Information regarding the time when the message was received and the driver ID is added to the message headers. Furthermore, the device id is used to retrieve the device type and order from the registered device database and added to the headers to simplify routing and application development.

3.3.2 Application Container

There are a number of platform that could be used as application containers. Docker would allow applications of any type to be deployed. Some language specific containers are also available for applications like Python and NodeJs. These are usually web-application centric, based on the Web Service Gateway Interface (WSGI). The container, which best fits the requirements as well as possesses extensive control of deployment and life-cycle management, was based on the Open Service Gateway Interface (OSGI) framework (Alliance 2003), which is designed for deploying modular java applications, dynamically on top of the Java VM. The Apache Karaf (Nierbeck et al. 2014) implementation of the framework has a number of add-on libraries that are key components in the development of applications using the Microservice architecture and in enabling a wide range of applications to be deployed side by side.

To allow applications to listen to specific events, the received messages are routed to the EventAdmin based on their headers and contained data. These routing rules can be seen in Table 3.2.

Table 3.2 OSGI Message Translation

Sender	Key Property	Receiver	Resulting Topic
device	dev_type	app	/device/receive/[dev_type]
app	*	device	/device/send/
Cloud	app_name	app	/Cloud/receive/[app_name]
app	*	Cloud	/Cloud/send/[app_name]
resource	resource_type	app	/resource/[res_type]/receive
app	resource_type	resource	/resource/res_type]/send
region	app	app	/region/receive/[app_name]
app	*	region	/region/send
app	app_name	*	/apps/[app_name]/send
*	app_name	app	/apps/[app_name]/receive

In order to allow for applications to respond to new devices being added to the system as well as to be able to listen to individual devices and have messages transmitted to these applications from the Cloud, they need to be able to reconfigure their application name and the name of the devices they are listening to. This is achieved through the ManagedService class's update() function that allows applications to read the configuration file. In this case, each device will have its own file where the gateway controller adds information regarding the devices the application is configured to communicate with as well as the application's name, the regional communication channel and other configuration parameters. When the

configuration is updated, all applications are refreshed to start with the new set of data. Applications can be deployed into multiple environments with multiple use cases to facilitate their testing and migration.

The applications can be managed through the gateway controller, while their status and performance are monitored through the bundles deployed on the container according to JSON based deployment files. For the migration of applications, the internal structures used for communication-like services allow for containers to migrate this service on the local region if configured properly. For the implementation, they are considered to be available only on the local deployment and applications linked through these services are required to be on the same gateway.

3.3.3 Regional and Cloud Drivers

The drivers used to connect to the Cloud providers are designed to broker messages from the local AMQP messaging service to a Message Queue Telemetry Transport (MQTT) server hosted on the Cloud. MQTT was chosen as the connection protocol to the Cloud due to the wide range support from major Cloud providers like AWS and the added functionalities these propose. This light-weight messaging format was designed for high-latency or unreliable networks so it offers the best solution for asynchronous messaging between Cloud and gateway. The Cloud communication drivers are designed to allow for single direction connections and can have multiple providers connected and routed through their instances.

Connecting to the Cloud can be done through the Secure Socket Layer (SSL) or through simple username and password authentication, depending on the security requirements and the provider's options. In order to send and receive information from the Cloud messaging service, the proposed broker translates the byte-array messages into headers and payloads as well as sending them to the required exchange. The first connection is to the main Cloud, which is done before the gateway starts. The other connections can be started through commands received on the first one using the gateway controller.

Messages on the local Cloud queue need to be parsed into a byte array and sent to the Cloud. The solution for this parsing problem is creating JSON strings from the received AMQP messages where each header and the payload are made into a JSON object. The payload is either parsed as one object, or as sub-components formatted in JSON as seen in Table 3.1. Drivers used for regional communication use REST APIs to receive and send messages from applications. Each application has the option of configuring one or more regional connections that can be used by outside applications. These are configured by creating a queue for each and routing the queues based on URL location on the REST APIs which get configured by the gateway controller. This configuration would allow applications

to have their own access keys and authentication option on the region. The other proposed drivers would rely on STOMP messages being routed to the messaging service based on correctly formatted headers. Messages with the appropriate configuration would be routed and those without the right data would be lost.

3.4 Distributed Control and Metering Use Case

The adopted use-case scenarios show a simple home and office monitoring and a control environment where the advantages that this system brings compared to other platforms are highlighted. The deployment scenario looks at cases where gateways are deployed in multiple rooms or environments having their own set of devices connected to them. This platform allows the deployment of applications to be done to a variety of gateway while making use of these resources from peers without the need to reprogram the applications. Furthermore, these applications can communicate with a wide variety of devices through the same format regardless of the devices' communication protocol. This allows for a better use of resources on the gateways as well as facilitates the deployment of complex applications. These functionalities would be very difficult or impossible to implement on the reviewed platforms.

The setup has the configuration of a smart home thermostat. It consists of a humidity sensor, temperature sensor, presence sensor and an actuator device that turns the heating on and off. The application reads data from the devices and controls the actuator based on a control algorithm while saving all relevant temperature readings to the database and all important events to the log. The applications are able to receive user commands locally or from the cloud. The setup also allows for the applications to be migrated among gateways and to the cloud based on the configuration needed. A deployment of the system can be seen in Fig. 3.5 where the first application is deployed on the cloud while the second one is deployed on the RF24 capable gateway.

The first application has the task of collecting the sensor data, saving it to the database and the log files and sending periodic reports to the second application. The second one is tasked with reading reports from the first application and comparing those to its control algorithm and sending control signals to the actuator, while saving information to the log files and sending reports to the second cloud connection. These two applications can be deployed anywhere on the two gateways or the cloud, being able to control and read the devices while performing the logging and storage tasks.

The three devices have their unique sensors and tasks, and are designed in a way that they can be analogous to low power sensors. The two sensing devices are Atmega Attiny85

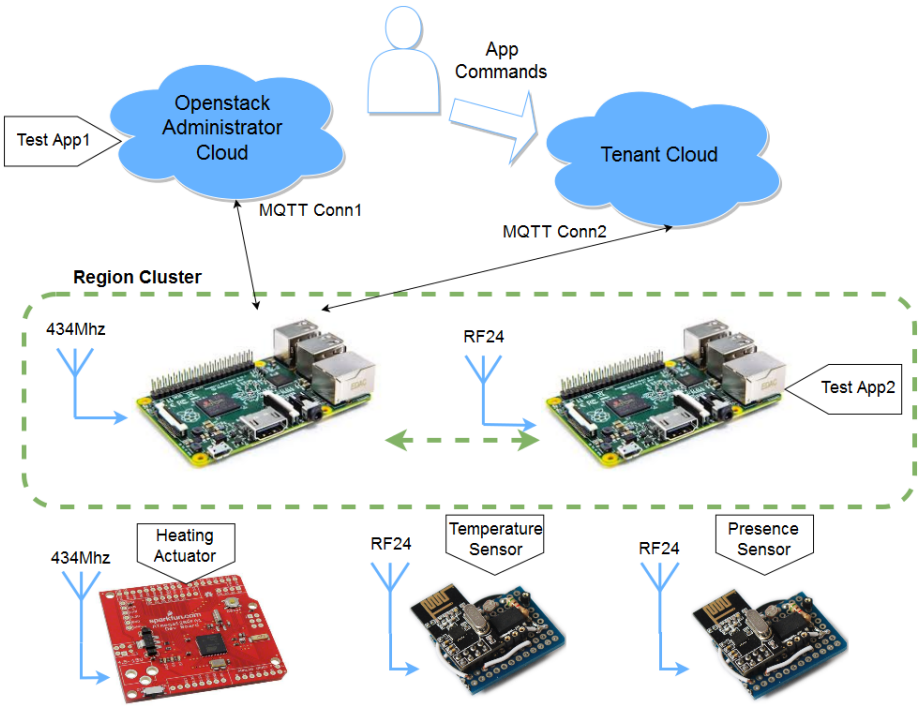

Fig. 3.5 Control and Metering Application

boards equipped with NRF24L01 transceivers, one of them with a temperature sensor and the other with a light based presence sensor. The actuator is an Atmega128rfa1 434MHz communication enabled micro-controller that signals a relay that controls the heating agent. These devices communicate with the gateways through the dedicated drivers. An NRF24L01 transceiver is connected to the GPIO pins of the Raspberry Pi. The heating actuator is connected to the first Raspberry Pi, while the temperature and presence sensors are connected to the second one.

Consider the route a set of messages take through the system to go from peripheral device to Test App1, Test App2 and then come back as an actuator action. The messages from the Temperature sensor connected to the Gateway housing the first application are sent to this application by first being processed by the RF24 Dongle Communication Driver from Fig. 3.1 which then routes the message through the Device Exchange to the Apps Queue as shown in Fig. 3.2. This message is then translated by the Messaging to Event Admin Broker to the Application Container. Here the Application reads the message and send the processed data to the logger, storage and the second application. A message coming from a different gateway to the same application would be routed in the same way, but with the Route between the Device Exchange and Apps Queue being extended by not routing the message to the local Apps Queue but sending it to the Region Resolver that sends the message

to the appropriate gateway queue, where the message is then sent through the Regional Communication component to the gateway hosting the application. Here, the message arrives through the appropriate Region Exchange and is Routed to the local Apps Queue, where it follows the previous route. This application saves data to the logger and database in the same way, but sending a message to the Event Admin in the Container that then brokers the message to the Messaging Services' Apps Queue that routes the message to the Resource Resolver that then puts the message in the right resource queue. Local Resource Access Drivers then take these messages and perform the corresponding actions. This application sends messages to it's control counterpart through the same system, but is this case the Apps Exchange Routes the message to the Region Resolver. It then sends the message through the Regional Communication component to the other gateways Region Exchange that then routes the message to the corresponding Apps Queue. The control application finally receives an event that looks exactly as if it were generated in the same container by the metering app. The control app adjusts the thermostat by sending a message through the Containers messaging broker to the Apps Exchange, which then gets routed to the Device Resolver and through the corresponding queue and Device driver to the device. Through these routing techniques, the two applications can communicate with each other and the devices as if they were on the same gateway, or in this case same virtual environment.

The implementation shows the clustering and migration functionality of our platform as well as scenarios where applications may need to communicate with devices connected to other gateways on the region and how this is deployed. Finally, we demonstrate that the presented implementation shows functionalities and scenarios which surpass the capabilities of other systems, due to migration and message passing between gateways as well as presenting a proposed use-case for the system.

3.5 Summary

The middleware platform and its components described in the previous sections are designed to satisfy the connectivity and resource management requirements of Industry 4.0 through message translation and routing. This research proposes extending these requirements with the concepts of Fog Computing where the system resources are virtualised and the users view the system as one homogeneous entity.

The messaging service and the translation of messages through the SenML and AMQP format are designed to produce a homogeneous view of a system that comprised heterogeneous sensors and devices that have varying methods of communication, reliability and latencies. This translation is also used to allow message transmission from a source gateway

to a destination one. The same method is used to translate inter-app or service messages to be transferred between application containers. This supports application migration and also allows the system to support a single system view.

This view and the translation methods add extra overhead to every message but is crucial if migration and message routing is to be considered. This in hand also shortens development times by allowing developers to focus on functionality rather than lower level connection programming and message sending. This system also allows for drivers to be developed and deployed once which increases horizontal integration of these services and since the used format is an industry standard it can be easily applied to allow multiple platform, Cloud and device collaboration.

The Platform was implemented to the full extent of the description, supporting an OSGI Karaf Based Application Container and Monitoring. This implementation can be downloaded from the GitHub Repository (*https://github.com/nandor1992/FogOfThings*). This implementation contains: drivers for RF24, 434Mhz, Xbee and Bluetooth peripheral drivers; Logging and CouchDB based database management components; MQTT to AMQP brokers for cloud connection; Local Administrator component that manages RabbitMQ, CouchDB and Karaf; Karaf based Container monitoring elements and Linux gateway monitoring components; Cloud controller that is able to receive the deployment JSON files and send the appropriate commands to the nodes; Example of an Android-based client application for the Temperature control use-case, as well as the code for the CNC control use-case. The*JavaUpdates* branch of the repository contains all the optimisation code that was used. This repository doesn't contain any deployment code that is designed to deploy the system, all the services and libraries need to be installed and the right configurations set up.

This system, while having numerous benefits also results in a number of challenges as deploying applications on the system in such a way that latency, reliability and other characteristics are maintained to assure QoS constraints and to satisfy SLA agreements becomes a very complex optimisation problem. Due to the highly shared environment in which applications are deployed, traditional VM based models are not suitable to model these platforms. The upcoming chapters look at modelling these parameters for individual applications, gateways and devices. Multiple parameters are taken into account and varying utilities are proposed.

The proposed Fog of Things Platform is shown to answer the requirements stated in Section 3.1 by highlighting the resulting capabilities in the use-case in the previous sections as well as through the extensive use-cases presented in Section 6.1.1. The platform also undergoes a quantitative analysis of its run-time capabilities in the analysis Section 6.1.2.

Chapter 4

Application and Gateway Model

4.1 Overview of Model

The application model attempts to estimate the functioning parameters of an application based on the limited information available, making estimation and optimisation possible. The model attempts to calculate the load of the application and its connected devices on the gateway which is used to estimate the effects of migration. To be able to measure the total delay of the device messages test drivers are used, that allows messages to be sent to the application container at a constant rate and allows the measurement of the actual return times of the messages. The processing delay is measured by the application itself. The delays between drivers and physical devices are not considered, because these delays cannot be improved by the system and are subject to the adopted protocols and underlying connections.

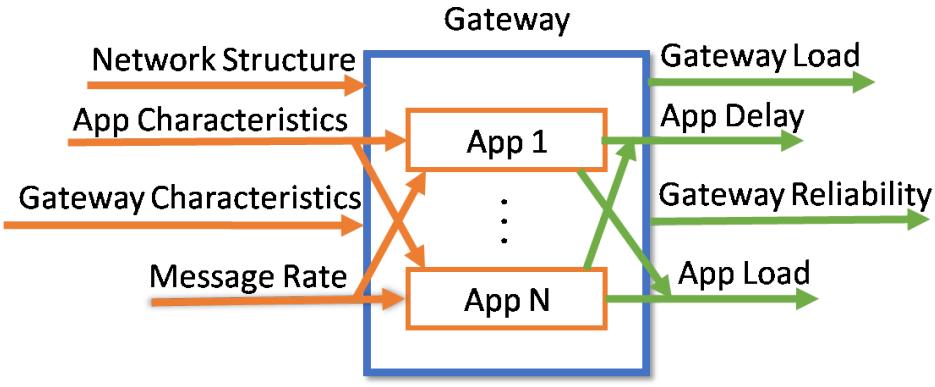

Fig. 4.1 Overview of Model

An overview of the approach that was used to defining the model can be seen in Fig. 4.1, where two distinct systems can be observed. The gateway and the application. Each

component has a set of inputs based on which it generates a set of outputs. The applications sit on top of gateways so they can be considered as a part of the gateway. The main task of the model is to take measured variables and constants and estimate variables that cannot be measured in normal application deployment.

The novel elements of this model consist of the identification of the application's characteristic Unit Load L^u_{ij} based on the measured load on the system as well as the characterisation of the gateway based on its processing capacity P^{Cap}_j and speed P^{Speed}_j. Another novel element in the model is the use of the network structure and individual connections and resources for the definition of loads and delays.

The elements of the model consist of three distinct types of variables. The measured variables such as the Measured Application Load L^a_{ij}, Network latency $D^{Ping}_{j,k}$ and others can be measured by the monitoring component. Variables such as the Time Delay Calculation constants $k1$ and $k2$ are characteristic to the system and are derived from experiments. Finally, variables such as the Total Application Delay D^A_{ij} are estimated, or more precisely, calculated by the model based on the other two variables.

4.2 Gateway Load

The gateway load is measured by the monitoring component on the gateway and is defined as the total CPU usage of the system in (%). The gateway has two types of overhead, the first type is generated by maintaining the cluster connections and background applications. The second is generated by device message processing by their respective drivers. Both are constant to a gateway and are not improved by migration.

The Gateway Cluster consists of the Gateways G_j having j denoting the gateway number containing Applications A_{ij} defined as application number i owned by gateway j.

To characterise a gateway or cloud node in this system a set of variables were defined to allow variability in the amount of processing a certain system can do and how fast this processing can be done. This distinction is important as with multi-core systems the processing capacity or quantity of processing a system can do increases, but the speed of how fast it can run a single thread is still tied to the speed of a single processor. To account for these the processing speedup P^{Speed}_j and processing capacity P^{Cap}_j parameters were introduced. The Gateway processing speedup P^{Speed}_j is calculated in Eq. (4.1) by comparing the execution time T^{Gw}_j of a bit of code on the gateway to the reference value T^{Gw}_{Ref} which was run on the node with a P^{Speed}_{Ref} of 1.

$$P_j^{Speed} = \frac{T_j^{Gw}}{T_{Ref}^{Gw} P_{Ref}^{Speed}} \tag{4.1}$$

The Gateway processing capacity P_j^{Cap} is calculated in Eq. (4.2) by comparing the measured application load of a known reference application and comparing it to the value measured on the reference gateway that has a processing capacity P_{Ref}^{Cap} of 1.

$$P_j^{Cap} = \frac{L_{ij}^A}{L_{iRef}^A P_{Ref}^{Cap}} \tag{4.2}$$

The Gateway Load L_j^{Gw} is the sum of Measured Application Loads L_{ij}^A and the Base Load L_j^B. The equation for the Gateway Load L_j^{Gw} can be seen in Eq. (4.3) and is measured in % of the CPU usage. This variable is directly measured through the monitoring component.

$$L_j^{Gw} = L_j^B + \sum_{1:N}^{i} L_{ij}^A \tag{4.3}$$

As defined in Eq. (4.4), the Base Load L_j^B is the processing power used by system/background processes and drivers which are considered constant throughout the migration process.

$$L_j^B = L_j^{Ide} + \sum_{1:N}^{k} L_k^D \lambda_{jk}^{Gw} * P_j^{Cap} \tag{4.4}$$

The Idle Load L_j^{Idle} in Eq. (4.4) is the % CPU of the gateway j at rest without any message passing or any application related activities. It is considered that this is constant on every gateway, regardless of applications or connected devices or peers.

The Gateways Driver Message Rate λ_{jk}^{Gw} used in Eq. (4.4) is defined as the total number of messages n_{jk} sent and received by driver k on gateway j in a certain time interval Δt and is measured in messages per second (msg/sec). The Driver Load L_k^D in Eq. (4.4) is described as the %CPU used by driver k to communicate with the devices connected to the gateway through the driver for a certain message rate and is specific to each device driver.

The power consumption of a server or gateway can be a factor of the CPU Load, IO Rates, Storage, Memory Accesses and other peripherals. Considering a case where through migration the resources and drivers are accessed from the same location, the main components that are of interest would be the CPU Load and added communication between gateways. If considering as this work does, that Reliability of the communication can be partially

accounted for by its effect on the CPU the work in (Blackburn and Grid 2008) is used, where the Power Consumption can be defined as a factor of the Gateway CPU.

4.3 Application Load

The total load of an application can be modelled based on the test data from the total CPU usage of the application threads and the known messaging overhead added by the driver and the container broker. These units are defined based on the test application set but would describe any measurable application deployed on the gateway that functions in a similar manner.

$$L^u_{ij} = \frac{L^a_{ij}}{\lambda^A_{ij}} P^{Cap}_j \tag{4.5}$$

The Application Unit Load L^u_{ij} is defined as the reference processing power used to process one message of the application regardless of the current Message Rate. The Unit Load in Eq. (4.5) is used to compare the behaviour of different types of applications based on their processor use and message rate, without knowing anything about how they work. Considering that the processing power of the host remains the same, the L^u_{ij} of the application is constant indifferent of deployment location. The unit load is measured in % of the CPU usage for the specified message rate or $(\%cpu/msg/sec)$

$$L^A_{ij} = \frac{(L^u_{ij} + L_M)\lambda^A_{ij}}{P^{Cap}_j} \tag{4.6}$$

The Total Application Load L^A_{ij} in Eq. (4.6) defines the weight of the application on the Gateway j which is defined as the sum of the application unit load L^u_{ij} and the Message Load L_M adjusted to the processing capacity coefficient P^{Cap}_j and multiplied with the application message rate λ^A_{ij}.

The Message Load L_M in Eq. (4.6) is a constant that denotes the processing impact of the received messages of an application and is the load created by the broker driver between the messaging service and the containers event service. L^a_{ij} and λ^A_{ij} in Eq. (4.5) are measured by the monitoring application while L^u_{ij} and L^A_{ij} are calculated.

The Application Message rate or λ^A_{ij} used in Eq. (4.6) is defined as the total number of messages n_{ij} sent and received and application i on gateway j in a certain time interval.

The Measured Application Load L^a_{ij} in Eq. (4.5) is defined by the amount of CPU the application threads are using on average when running and is denoted by % of the total CPU, as measured by the container monitor.

4.4 Delay Model

The delay of the application can be modelled based on the load of the gateway they are on, the number of messages they receive and the amount of load they generate on the gateway. The model limits its predictability to applications that only perform major processing tasks based on device messages and do not perform background operations. This method can be used for a general application set but its predictability decreases by the amount that the test application differs from the test model.

The Application Delay D^A_{ij} can be formulated in Eq. (4.7) as the sum of all application related delays, and is the sum of the processing delay D^P_{ij} and the networking delay D^N_{ij}.

$$D^A_{ij} = D^P_{ij} + D^N_{ij} \tag{4.7}$$

The Processing Delay D^P_{ij} in Eq. (4.8) is deduced from the experimental data in the parameter analysis section, where it is a function of the Gateway Load L^{Gw}_j in Eq. (4.3), the Unit Load L^u_i defined in Eq. (4.5), the Gateway processing speedup coefficient P^{Speed}_j and the two constants $k1$ and $k2$. The values of $k1$ and $k2$ are analogous to a reference IPS^{Gw} value and how it changes with load. The value of $k1$ defines the reference IPS rate of the gateway with no load while $k2$ defines how the $k1$ value is reduced with added load. These reference values are adjusted to differentiate between gateway processing capabilities using P^{Speed}_j.

$$D^P_{ij} = L^u_i \; P^{Speed}_j \; (k1 + k2 \; L^{Gw}_j) \tag{4.8}$$

The Networking Delay D^N_{ij} can be described by the sum of delays generated by routing messages from one gateway to another through the messaging system and is described in Eq. (4.9).

$$D^N_{ij} = D^R_{Base} + \sum_{k=G_{Migrated}}^{j=G_{Host}} D^{Ping}_{j,k} + D^R_{Ext} \tag{4.9}$$

$D^{Ping}_{j,k}$ represents the network latency between j and k while the sum is the set of delays that are required to link two gateways. If the application is deployed on the same gateway as the devices, then $D^N_{ij} = D^R_{Base}$.

The External Routing Delay D^R_{Ext} in Eq. (4.9) is a component of the Networking Delay and is an experimentally derived platform specific constant that denotes the average time it takes for a message to be routed from one gateway to another, not taking the latency inside the network into account. The Base Routing Delay D^R_{Base} is the delay added to any message

going from the driver to either local or external application, and is a fixed experimental and platform specific value.

The Ping Delay $D_{j,k}^{Ping}$ in Eq. (4.9) represents the latency between two directly connected gateways and is measured by the monitoring component on the gateway.

$$D_{Tot} = \sum_{j=1}^{N} \sum_{i=1}^{M} D_{ij}^{A} \tag{4.10}$$

The Overall Delay of the system D_{Tot} is described in Eq. (4.10) as the sum of all application Delays defined in Eq. (4.7) on the system.

4.5 Reliability Model

For the reliability model, only the influence of the Total CPU load on the Reliability of the system are considered, because driver message rates and access values are unchanged when migrating. Furthermore, it is considered that the decreased reliability due to added message rates between gateways is covered by the increased Gateway Load $L^{G}w_{j}$.

The load-hazard model in (Iyer and Rossetti 1986) is adapted, which provides an analysis of the impact of Load on the probability of the machine to run without any CPU or system errors. They propose a load-dependent hazard or failure rate model $z(x)$, where x is the gateway load L_j^{Gw} . They prove statistically that there is a correlation between increased load and increased failure rates. They define the hazard rate $z(x)$ as the probability of an error occurring at a CPU load $x+\Delta(x)$, where $\Delta(x)$ is the added load, given that there was no failure at load x.

Considering Eq. (4.11) for defining reliability, where $R(t)$ is a function of the hazard function $z(t)$ or a function of the constant failure rate λ. In the case of the hazard function λ can be replaced with $z(L_j^{Gw})$ as it is considered constant in time.

$$R(t) = e^{-\int_0^t z(u)du} = e^{-\lambda t} = e^{-z(L_{Gw})t} \tag{4.11}$$

Based on the data from (Iyer and Rossetti 1986) the hazard function $z(L_j^{Gw})$ is approximated to a third-degree polynomial Eq. (4.12) where x is between 0.08 and 0.96 in CPU usage, the min value is 0.0018 and the max value 0.0118

$$z(x) = 0.0195x^3 - 0.137x^2 + 0.0059x + 0.0015 \tag{4.12}$$

Considering a constant run-time of a day or 24 hours, the Gateway Reliability R_j^{Gw} can be defined using the Load based Reliability function $R(L_j^{Gw})$ in Eq. (4.13). This results in a maximum reliability of 95.5% and a minimum reliability of 75.35%.

$$R_j^{Gw} = e^{-0.468{L_j^{Gw}}^3+0.3288{L_j^{Gw}}^2-0.1416L_j^{Gw}-0.036} \tag{4.13}$$

Based on the gateway reliability in Eq. (4.13), the application reliability R_i^A can be calculated as the product of all the gateway reliabilities where the application has resources deployed. This can be seen in Eg. 4.14 where k represents a resource and peer applications of the tested application i deployed on gateway j. R_{kl}^{Res} represents the Reliability of the Gateway l where the resource or peer k resides. It is worth noting that k represents a unique gateway, so if resource k_1 and k_2 reside on the same gateway, the reliability of this gateway is only considered once in the product.

$$R_{ij}^A = R_j^{Gw} \prod^{k \in A_i} R_{kl}^{Res}, where\ \mathbf{l}\ is\ unique \tag{4.14}$$

4.6 Parameter Analysis

The tests are performed in a physical environment based on the scenario presented in (Verba, Chao, A. James, Lewandowski, et al. 2017) and in the Use-Case of the Evaluation section. This configuration is used to find the parameters of the application and gateway model. The testing environment consists of homogeneous Raspberry Pi nodes that have a processing speedup coefficient of P_j^{Speed} and processing capacity coefficient P_j^{Cap} of 1. When migrating to the Cloud, there are two VM's on different hosts with varying processing capabilities.

4.6.1 Processing Capacity and Speedup

The Gateway processing speedup P_j^{Speed} and processing capacity P_j^{Cap} are calculated in Eq. (4.1) and Eq. (4.2) by deploying a reference application for five minutes and measuring a stable four minutes the load it creates and delays of the application. For the tests, a set of 5 applications with varying loads on top of four types of gateways are deployed on the two VM's and two Raspberry Pi's. The first VM (VM1) has two VCPU's and 4GB of RAM on a host with an i5-Intel Xeon E312 3.1Ghz Processor with a BogoMIPS value of 6185.94 . The second VM (VM2) has two VCPU's and 4GB of RAM on a host with an i7-4770 3.4GHz Processor with a BogoMIPS value of 6784.28. The Raspberry Pi 2 model B has an ARM Cortex-A53 1.2 GHz quad-core processor and 1GB of ram. Two scenarios are considered with the Raspberry pi, for the first (RPi1) is overclocked to 1.2GHz and for the second (RPi2)

is left at the base value of 1GHz. This overclocking changes the BogoMIPS value of the pi from 697.95 to 732.2. The more performant VM and Raspbery Pi is used for the rest of the testing and evaluation.

For the estimation of the processing capacity P_j^{Cap} values, the systems CPU use is measured when on idle resulting in an L_{Idle} value of 3.68% for the Raspbery Pi's and 1.8% for the VMs. After this, the load application with varying loads is deployed. This application mimics the standard applications but it performs the processing tasks on a timer rather then on received messages making the created load more stable. The deployed loads were 0, 50, 100, 200, 300, 500, and 1000 cycles of the processing task. The CPU load of the Machines is measured and based on Eq. (4.2) the processing capacity of the machines is calculated.

For the estimation of the processing speedup P_j^{Speed} an application is deployed on the machines and its Processing Delay D_{ij}^P is measured. The application deployed was given 50, 100, 250, 500 and 700 cycles of the processing task with a message rate of 5 giving the unit load L_{ij}^u values of 0.55, 1.008, 2.448, 4.76 and 6.6126. The processing times of these applications are then measured and based on Eq. (4.1) the processing speedup of the machines can be calculated.

Table 4.1 Processing Parameters of the Machines

Machine	VM1	VM2	RPi 1	RPi2
Processor	I5-Intex Xeon 3.1 Ghz	i7-4770 3.4 GH	ARM Cortex-A53 1.2 GHz	ARM Cortex-A53 1.0 GHz
CPU Count	2	2	4	4
BogoMips	6185.94	6784.28	732.2	697.95
P_j^{Cap}	2.304	2.556	1	0.924
P_j^{Speed}	4.189	4.307	1	0.874

The results of these experiments can be seen in Table 4.1., where the differences between the machines can be seen. Here,the BogoMips is a good indicator of the relation between two gateways processing coefficients, it however cannot be used to estimate these values.

4.6.2 Driver and Message Loads

The loads of the drivers L_k^D are calculated on the system by measuring the idle load of the system with the drivers running but no messages being sent through, after which messages at

different rates are sent to a non-routable queue and the added load on the CPU is measured. The results are divided with the message rates to get the driver characteristic load for each message received. This was tested for the RF24 , Bluetooth and the TestingDrivers. The tested message rates were 1, 3, 5, 10 and 20 messages every second.

To measure the load of routing the messages on the system and through the Karaf container, the TestingDriver application is used to send messages to an application with a known Unit load $L_i^u j$ of 1.008 with varying message rates of 1, 3, 5 and 10. The application load is then measured and the base load L_j^B is calculated based on the message rates and the initial idle values. Based on Eq. (4.6), the value of L_M is then calculated.The mean value of the four tests is considered as the reference value.

For the Driver Loads L_k^D an averages is measured of 1.61 for each RF24 message, 0.73 for each Bluetooth Message and 1.10 for the Testing Driver. The difference in created load can be attributed to the implementation of the drivers but also the hardware support on the system. While the Bluetooth driver has a dedicated chip that takes off some of the load by implementing more of the OSI model on chip, the RF24 most of this needs to be done by the driver. For the driver, the load is caused by the measurements and data retention. The average messaging load L_M on the system was found to be 0.285.

4.6.3 Processing Delays

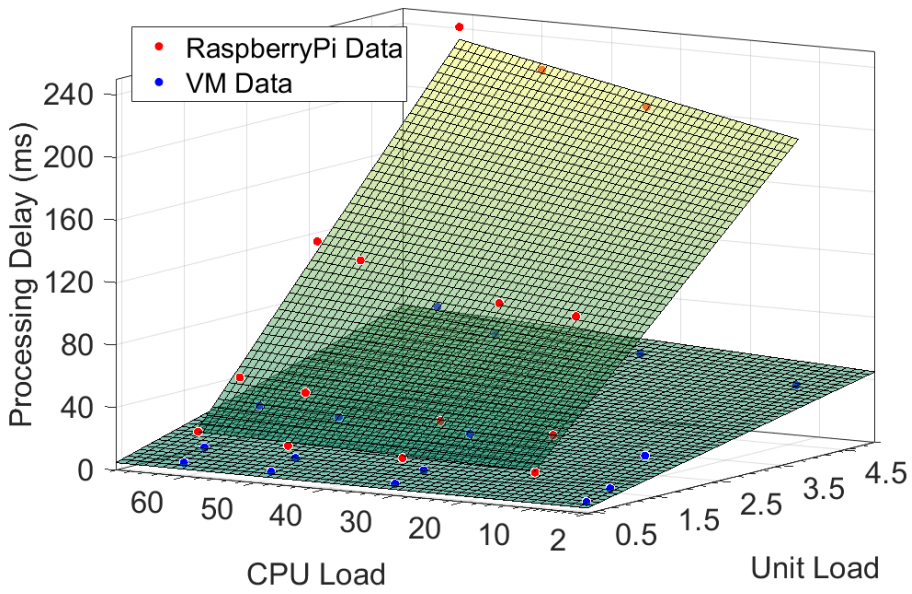

Fig. 4.2 Processing Delay Variation

To estimate the processing delays D_{ij}^{P} of the system k1 and k2 need to be determined, which represent the characteristic delay for a certain amount of processing done with a certain amount of load on the system. From the tests, it can be concluded that the RAM of the system only affects the delays when the system goes above 80% ram use, in which case the system may crash. Furthermore, the message rate only influences every individual message by adding load to the system, and when that contribution is taken out no further delay is added. If the message rate exceeds the Karaf driver capacity or the application capacity and overload situation is considered, as the Karaf driver causes a bottleneck. In conclusion, for the system the impact of the Gateway Load L_{j}^{Gw} and the Application Unit Load L_{ij}^{u} is tested within normal operation parameters of the system where RAM use does not exceed 80% and the messaging rate does not cause bottlenecks. Testing for these is done through the Gateway monitoring application for RAM and the RabbitMQ monitoring for queued up messages to validating the message rate.

For the testing, four known applications are deployed with unit loads of 0.55, 1.008, 1.906 and 4.76, and a constant messaging rate of five messages every second. The total CPU utilisation of the system and the Application load was measured through the monitoring component. To account for different scenarios, the CPU usage of the gateways were increased through the Load app. These scenario were deployed on VM1 and RPi1 and got the points presented in Fig. 4.2. Using the equation from Eq. (4.8) and the performance values from Table 4.1 a curve fitting is done to match the data-points, having k1 and k2 as unknowns. The resulting function provides the slightly curved form of the two surfaces.

$$\begin{aligned} D_{ij}^{P} &= L_{ij}^{u}\, P_{j}^{Speed}\, (38.409 + 0.1885\, L_{j}^{Gw}) \\ D_{i\,VM1}^{P} &= L_{ij}^{u}\, (9.169 + 0.0459\, L_{VM1}^{Gw}) \\ D_{i\,RPi1}^{P} &= L_{ij}^{u}\, (38.409 + 0.1885\, L_{RPi1}^{Gw}) \end{aligned} \tag{4.15}$$

The resulting equations for D_{ij}^{P} in the general case, for VM1 and for RPi1 can be seen in Eq. (4.15). The value for k1 is 38.409 and 0.1885 for k2 where the first means that for each unit of load, 38.409 milliseconds of processing on a reference system needs to be added, while k2 means that for each percentage of extra load 0.1885 milliseconds of processing delay for each unit of load on the reference system is added. The Processing Delay Constants fir RPi1 stays the same, at 38.409 for $k1$ and 0.1885 for $k2$. The values for VM1 are 9.169 for $k1$ and 0.0459 for $k2$.

4.6.4 Networking Delays

The Networking Delay is measured by testing the response time of an application when running on the RPi1 and when running on a VM1. The monitoring component measures the ping values between peers which allows the system to compare response times to the ping values. The latency between the two gateways was increased using netem (Hemminger 2005) for Linux from 0 to 80ms to match typical Cloud-user latencies.

$$D_{ij}^{N} = 9.83 + \sum_{k=G_{Migrated}}^{j=G_{Host}} D_{j,k}^{Ping} + 8.246 \tag{4.16}$$

To get the value of the base Routing Delay D_{Base}^{R} and that of the External Routing Delay D_{Ext}^{R} , the processing delay is subtracted from the total delay which results in the total D_{ij}^{N}. From this, measured ping value $D_{j,k}^{Ping}$ between the two gateways is subtracted which results in $D_{Base}^{R} + D_{Ext}^{R}$. The mean values for local processing where only D_{base}^{R} is present with a value of 9.83ms is then subtracted from equation resulting in D_{Ext}^{R}, that has a mean of 8.246ms giving rise to the equation for D_{ij}^{N} in Eq. (4.16).

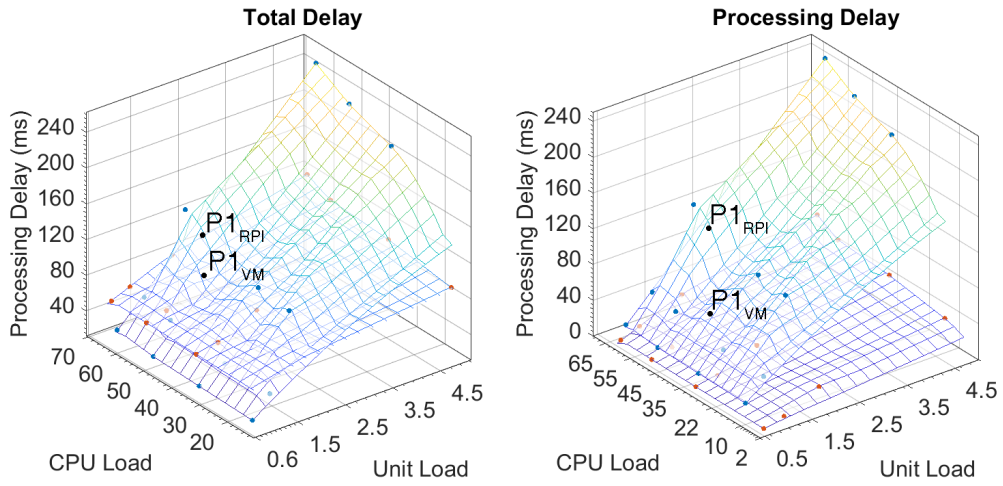

Fig. 4.3 Application Delays variation

The graphs in Fig. 4.3 shows the impact the networking delay has on the total delay, resulting in a set of gateway load L_{j}^{Gw} and Unit Load L_{ij}^{u} threshold values where applications have smaller latencies on the gateway than on the Cloud VM. Considering the P1 points on the figure, a difference in their values between processing and total delay on both the VM and the RPi can be seen. For these cases, the points are a result of having an app with a L_{ij}^{U} of 1.906 and a gateway load $L^{G}w_{j}$ for the RPi of 51.917% and 51.425% for the VM. This results in a calculated Total Delay of 101.69ms for the RPi and 90.57ms for the VM. The

actual values were 108.743ms for the RPi and 85.051ms for the VM. This results in a mean error of 6.29%.

4.7 Utility Functions

Providing a utility function for a system is a crucial part of its development as it provides a way for developers to improve or analyse changes in a system. This utility function needs to represent the needs and health of individuals in the system but should also provide an overview of how the system is performing as a whole. In the case of the model this utility function is used to evaluate a set of multi-variable characteristics of the system and how they affect individual applications and the system as a whole.

When compiling the parameters that should comprise the utility function these need to be quantifiable based on the model that is proposed and these need to be relevant to the system. This is the reason why even though the load of gateways is calculated it is not considered a relevant component and why even though through Green Computing the energy consumption of the system is important, based on the used model, it is impossible to estimate the value of this component so it is left out. The parameters that are considered are based on (Verba, Chao, A. James, Lewandowski, et al. 2017) where some key components and parameters of some industry-based systems are identified.

When considering the general utility function, it can be decomposed into three components: delay, reliability and constraint violations. To consider a utility function that can be as generic as possible, each application is considered to have different preferences to reliability and delays as well as constraints. To account for this, individual weights for each component are defined, where W_i^X denotes the weight of an application $A_i \in A$ for the parameter X where $X \in \{Delay, Reliability, Constraint\_Violations\}$

The first component focuses on the Delay of each individual application D_{ij}^A where the delay component D^F can be defined as in Eq. 4.17 as the sum of all application delays compared to their reference status multiplied by the individual weight for those applications. The normalisation of the delays to each applications' reference position puts the applications in an even playing field where large applications do not have an unfair advantage or priority. This also helps provide the utility component of the system, that is going to have a rough average value around or lower than 1.0. Considering D_{iRef}^A as in Eq. 4.18 where the Total Delay of the application D_{ij}^A is considered in a case where the Networking component D_i^N is 0.0 and the Processing Delay D_i^P is calculated for the Average Gateway load and a speedup of 1.0. The average load of the system can be calculated by looking at all the adjusted

Processing capacity available on all the gateways and extracting all the required processing capacity by the applications which does not consider networking requirements.

$$D^F = \sum_{i=0}^{n} D_i^F = \sum_{i=0}^{n} \frac{D_{iRef}^A}{D_{ij}^A} W_i^{Delay} \tag{4.17}$$

$$D_{iRef}^A = L_i^u (k1 + k2 \frac{\sum_{j=0}^{m} L_j^{Gw}}{m}) \tag{4.18}$$

The second component focuses on the Reliability of applications on the system and can be defined as R^F as in Eq. 4.19 as the sum of all application Reliabilities R_{ij}^A compared to the reference system reliability. As with the previous component, the normalisation provides a view for each individual as compared to itself rather than its absolute value. The reference value calculation can be seen in Eq. 4.20 where the average utility of gateways is considered as a proxy to the average utility of Applications as this might not be known.

$$R^F = \sum_{i=0}^{n} R_i^F = \sum_{i=0}^{n} \frac{R_{Ref}}{R_{ij}^A} W_i^{Reliability} \tag{4.19}$$

$$R_{Ref} = \frac{\sum_{j=0}^{m} R_j^{Gw}}{m} \tag{4.20}$$

The third component of the utility function represents the constraint violations of the application deployments Ct^F. This can be seen in Eq. 4.21 where the individual constraint values Ct_i^{Delay} and $Ct_i^{Reliability}$ are considered. In the case of this formula, $+$ is considered as a numeric addition where the Boolean result of the inequality can either be 0 or 1. In this case the possible values for any individual Ct^F in the case of two components are $\{0, 0.5, 1\}$.

$$Ct^F = \sum_{i=0}^{n} Ct_i^F = \sum_{i=0}^{n} \frac{R_{ij}^A < Ct_i^R + D_{ij}^A < Ct_i^D}{2} W_i^{Constraint\_Violations} \tag{4.21}$$

When the individual components of the utility function are compiled, the resulting global utility function $Util^F$ takes the form as in Eq. 4.22. This represents the sum of all the components. The problem with this utility function is that it automatically scales with the increase of the Fog system, so in order to scale the results Eq. 4.23 is used where the global value is divided by the total number of apps resulting in the mean utility $\overline{Util^F}$. This allows for the comparison of systems of varying sizes.

$$Util^F = D^F + R^F + Ct^F \tag{4.22}$$

$$\overline{Util^F} = \frac{D^F + R^F + Ct^F}{n} \tag{4.23}$$

When deploying large systems or only partial systems as it is in the case of clustered deployments not all the applications are deployed so $Util^{F\,\prime}$ is defined as the sum of local or clustered partial utilities $Util_i^{C\,\prime}$. The actual local utilities can then be calculated when all the components are deployed resulting in $Util_i^C$. Subsequently, these result in the formulas for the cluster utility and the adjusted cluster utility in Eq. 4.24. Here a set of Clusters C_i are considered, where $i \in C$ and n_i^C represents the size of cluster i.

$$Util_i^C = \sum^{i \in C} D_i^F + R_i^F + Ct_i^F \qquad \overline{Util_i^C} = \frac{\sum^{i \in C} D_i^F + R_i^F + Ct_i^F}{n_i^C} \tag{4.24}$$

When considering highly heterogeneous multi-environment deployment, it can be considered that certain nodes or gateways will have platform and system capabilities Cap_j^{Gw} that are unique to a subset of these nodes. Furthermore, it can be considered that some applications have requirements for these capabilities Cap_i^A where they could not be deployed in systems where these capabilities are not met. This component needs to be considered on validation where an application is only allowed to be deployed on a gateway if the gateway can satisfy all of the capability requirements of the application. Not satisfying this condition means that a deployment is not viable and thus its utility function $Util^F$ can be considered 0.0. To differentiate between scenarios that cannot be deployed, the deployments are differentiated based on the number of capability and gateway max load violations that occur where less is better.

4.8 Summary

When considering CPS and Industrial environments, the latencies of a system and its reliability are crucial components. Estimation and optimisation attempts in Fog Computing and IoT need to consider as complete a model as possible for their methods, so they do not lose applicability and accuracy.

The model and platform provide a way of measuring and estimating the run-time parameters and migration benefits of applications in such systems. Based on the model, a set of novel approaches are proposed for load and delay optimisation through application migration between the Edge and the Cloud. These allow the estimation and improvement of certain parameters of deployed applications. Inspired by (Kunz 1991), an experimental load model description derived from measuring run-time parameters over physical systems has been

developed and used to represent the gateway and application loads, which provide a more realistic estimation than theoretical ones presented in other papers.

The presented model and utility function serves as a base for the upcoming chapter, where the focus is on methods for improving the system and individual health of applications. Variations of the presented utility function are proposed, that have simpler scenarios as discusses in several papers as well as more advanced cases where the traditional method may not be able to satisfy all constraints and find a solution.

Chapter 5

Deployment Optimisation

5.1 Introduction

When considering the deployment optimisation of applications in any ICT environment it is crucial to identify the characteristics and requirements of the systems that come into play. Optimisation methods designed to work in Cloud environments might not be suitable for Edge or mixed Fog Environments. Furthermore, methods developed for VM-based load balancing, estimation and optimisation might neglect certain characteristics and issues that Container-based systems might have. Container based systems also neglect some of the issues and drawbacks of shared environments. These shared environments consist of applications being deployed on the same system, sharing processing, storage and RAM, as opposed to VM and Docker containers, where resources are allocated solely to an application.

The presented models and methods might have certain optimisation scopes in mind when they are being developed, which could be in line with Green Computing trends as in (Zeng, Gu, and Yao 2018), classical Load Balancing and Optimal Resource Utilisation approaches as in (Congjie Wang et al. 2017) or they might explore more advanced methods as Location aware deployments and consider communication delays between peers, clusters and Cloud.

In this chapter, different optimisation techniques will be presented that aim to solve certain issues concerning the deployment of applications in large-scale heterogeneous IoT environments. Some of these are designed to solve Execution time based issues such as the Random Clustering and Allocation method while more advanced methods are designed to improve on existing solutions by analysing partial or full results and explore improvement opportunities through more meaningful search space reduction through clustering and resource allocation.

5.2 Problem Description and Categorisation

The problem of allocating Application or Services to Gateways, Machines or VMs has been explored in (S. Kim, C. Kim, and JongWon Kim 2017). When considering the Off-Line approach to deployment optimisation rather than a load-balancing, on-line or on-the-fly approach to deployment, the number of possible methods that can be applied increases as the constraints on decision time are less stringent. More complete models can be used to determine the health of systems and more advanced goals can be set. The optimisation tasks increase in size when deploying hundreds of tasks, services or applications onto hosts that not only fit their needs or constraints but also work towards improving the global health or utility of the system.

The use of more advanced Application and Gateway models as described in Chapter 4 has the downside of adding extra complexity to the deployments. This could be the main reason why some proposals as (N. Wang et al. 2017) have used more simplified and less interconnected models. This is also a reason why using VM or Container based solution may seem more appropriate in certain situations as resources are fully allocated to one container or VM and varying response times and reliabilities do not occur or are reduced when other applications are modified or deployed.

Due to the high interconnectivity of certain components, there are several difficulties and limitations to the proposed methods. The main one with the most impact to these methods is the highly interconnected nature of applications with a varying number of gateways, peers, resources and edge connections. This increased interconnectivity means that the deployed elements can be considered as a graph and simply separating the environment into the smallest number of connected graphs will not solve the problem. The size of these will still be too large for certain methods and would not cause great improvement to this system. Because of this, calculating the exact utility of certain sub-deployments is impossible to do without having the whole system deployed. To account for this, the partial utility $\overline{Util_i^A}$ is defined for apps that represents for the partial utility of the app A_{ij} where only a set of known components are deployed.

A positive side to the high interconnectivity of these components is the possibility of creating clusters or cooperative sets based on these connections and their impact on the individual and global utility, which may produce better results. A part of the presented methods focus on finding these meaningful connections and reducing the search-space to a solvable amount. They achieve this by creating a hierarchy of connections based on custom distance measurements and disregarding less important connections.

The allocation problem of assigning the highly connected application to a set of gateways can be considered as a set of Applications $A_i \in A$ where $i \in 1, ..., N$, and Resources (Peripheral

Devices, Storage, Logging, Regional Connections, etc.) $R_j \in R$ where $j \in 1,...,M$ assigned to a set of Gateways G_k where $k \in 1,...,P$ where S_n denotes the possible set of permutations $\phi : N,R- > M$ resulting in the Utility function presented in Chapter 4. A case can be considered, where the Applications have at least one external connection that influences their utility. In this case the optimisation problem can be simplified so that it is analogous to the Quadratic Assignment Problem in Eq. 5.1 or simplified further to be similar to the Linear Assignment problem (LAP) presented in (Lawler 1963). Here, the Gateways are analogous to the Locations and the application are analogous to the Facilities.

$$\sum_{i,j=1}^{N} \sum_{p,q=1}^{M} C_{ijpq} X_{ij} X_{pq} \tag{5.1}$$

In this case, a non-zero interconnected utility function is considered for the applications, where Application and Resources are considered as Entities $A \cup R \in E$. The Koopmans-Beckmann Formulation (Burkard et al. 1998) from Eq. 5.2 is a special case of the generic formula presented in Eq. 5.1. Starting from the notation from Eq. 5.2 the Application and Gateway utility function can be presented in a special simplified case as equivalent to this equation. Here f_{ij} represents the number and value of interdependent parameters between application i and j as analogous to the flow between facility i and facility j. The latency, load or the change in reliability between gateways k and l, as hosts of application i and j can also be considered as analogous to the distance between locations $d_{kl} = d_{\phi(i)\phi(j)}$ where $\phi(i)$ is the gateway hosting application i. The utility cost of placing application i on gateway k is analogous to $b_{ik} = b_{i\phi(i)}$ which represents the costs of placing facility i at location k.

$$\min_{\phi \in S_n} \sum_{i=1}^{N} \sum_{j=1}^{N} f_{ij} d_{\phi(i)\phi(j)} + \sum_{i=1}^{N} b_{i\phi(i)} \tag{5.2}$$

Considering these analogies, it can be assumed that the optimisation problem can be broken down in a special case where the impact of other applications on the gateway load and thus the utility is not considered when computing b_{ik}. These make the problem more difficult to solve than the traditional Quadratic Assignment problem. The QAP problem has been proven to be an $NP-Comple$ and $NP-hard$ problem so when considering the QAP problem as a special case of this allocation problem, it can be concluded, that this optimisation problem is both $NP-Complete$ and $NP-Hard$.

Finally, due to the nature of the optimisation being an allocation problem most optimisation methods that rely on climbing a hill or a look for a valley would not work on such a problem as the allocation of resources lacks these characteristics. Finally, due to the highly interconnected nature of the problem, a ripple effect can be seen even with the single

movement switching operations between placements where even applications that have no connections with these are affected and their utility changes.

Due to the scale of future industrial setting and thus Fog and IoT deployments in these scenarios the requirements for Distributed solutions to solve the deployment problem are crucial. One of the major disadvantages of Global optimisation problems it that with the scale that comes with these problems and the NP-Hard nature and non-polynomial complexity, methods need to be proposed that reduce the search-space or put constraints and bounds to the system in such a way that it remains solvable within a reasonable time frame.

In the upcoming sections, the methods will highlight attempts at randomly reducing the search space and the size of problems as well as looking at methods of meaningfully reducing these while considering the characteristics of these systems and how they work. This second step also attempts to solve the problem of platform lock-in by considering applications have requirements and that gateways have a certain set of capabilities. A random clustering and allocation might leave the system in a state where no viable solution can be found.

5.3 Overview of Approaches

The approaches used to try and solve the above-mentioned problem range from a black box approach used through the Global Genetic Algorithms Optimisation to the Tuning and Weights based Clustering method that is proposed. These methods are combined, compared and tested in different environments to show their advantages, disadvantages given certain models and methods. A high-level view of the proposed methods can be seen in Fig. 5.1. Here, *a.* represents the modified global GA approach, while *b.* looks at the app to app connection and resource-based allocation as well as the random allocation in the case of weights values of 0. The direction *c.* and *d.* showcase the proposed methods components which are either initialised through sampling or using some initial weights

The graph-based analysis of the connections and distances of applications and their deployments on gateways is central to the proposed methods and their rationale. With the increase in size, a reduced improvement in utility when deploying systems through GA and an exponential increase in processing time can be seen on these systems. This warrants the need to break up the problem into smaller pieces. The random allocation and clustering proposes the most rudimentary methods of doing this but shows that in some cases even this method outperforms the global GA just because it can dig deeper when it comes to the outcome. The same can be said with respect to the allocation methods where not sharing the gateway between clusters improves the results just by removing load constraints.

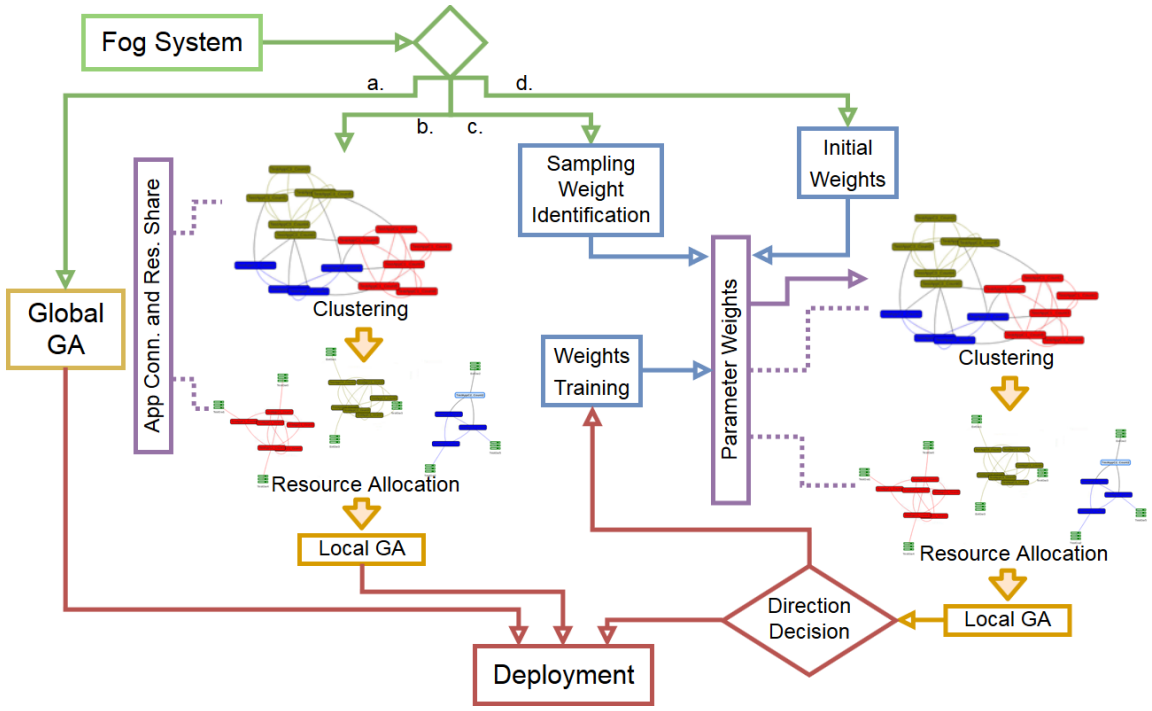

Fig. 5.1 High level view of Methods

The connection-based resource and application clustering attempts to remove some of the problems that are inherent with the random allocation by choosing peers based on their connections and by allocating gateways based on resources and access. This method maintains the requirements of the previous one with the allocation method maintaining the integrity of gateways and the cluster sizes being considered within reasonable values based on the previous test.

The training and weights based clustering and resource allocation method looks at providing a generic solution for application allocation on gateways with varying utility functions and scenarios with the condition that the parameters of gateways and applications need to have a relation with the output of the utility function. This method aims to analyse existing solutions and to reinforce good directions in clustering and allocations through a custom weighted distance measurement. The training of these weights falls into the category of Costly Global optimisation or Expensive Black-Box problems due to the difficulties in finding the global $Util_{Fog}$ from the set of local utilities $\sum_{i=1}^{N} \overline{Util_{Clust_i}}$.

To be able to account for a varying number of models and utility function a step forward from application connections and resource locations needs to be done when clustering and allocating resources. For these, a custom distance function is needed to account which parameters contribute to two applications collaborating and residing on a certain gateway. A set of best solutions is used and correlation calculation based on (Pearson 1895) are applied

to measure these. The clustering is then attempted using an extended DBSCAN (Ester et al. 1996) method that uses elements from OPTICS (Ankerst et al. 1999) to allocate apps that behave like noise. The issue with this method can be the overconfidence of certain directions and some high correlation number that may only be noise. The iterative method aims to solve some of these problems. This method can also work similarly to a random allocation method when false parameters are considered as important due to bad initial solutions.

Each method builds on the previous one, as all methods employ the GA method that is designed to attempt to allocate the applications inside the clusters onto the gateways. The connection and resource share based allocation builds on the random method as it attempts to find more meaningful clusters and allocate better gateways in a fair manner. The final solution then attempts to account for a larger set of models where other characteristics might determine whether two applications collaborate or are deployed to a certain gateway.

5.4 Deployment validation and Utility Calculation

The deployments that are generated using the global and Clustered GA method can sometimes allocate more application to gateways that the gateway has capacity for or more than the fair share of the Cluster. Furthermore, when considering application Requirements and gateway capabilities, whether a gateway can actually house the apps needs to be considered. The validation is done by clearing previous deployments, then deploying the apps to the gateways and calculating the resulting gateway load for each gateway of interest. If this load is larger than 99% for Global Clustering or larger than the Base Load plus the cluster share of the gateway then the validation has failed.

With small deployments and those that do not consider requirements this is not an issue, but producing valid individuals becomes an increasingly challenging task when considering large-scale deployments. Requirements based deployments put even more strain on the system and this issue becomes more prevalent.

5.5 Modified Genetic Algorithm based Method

Genetic Algorithms (GA) are evolutionary and meta-heuristic optimisation methods that are based on Darwin's theory of evolution. These typically rely on four basic operations to improve generations. These are inheritance, mutation, selection, and crossover. GA also has the advantage over other traditional optimisation methods that the concept behind it is easy to understand and to modify the parameters involved (Hoque et al. 2017). It is also able to avoid being trapped in minimum points by and employs a probabilistic selection

rule rather than a deterministic one. It has been previously shown that such methods work better than Hungarian methods given a random initial deployment in most cases as in (Verba, Chao, A. James, Goldsmith, et al. n.d.). This method allows for the use of a single Utility function through which local and global optimal solutions can be reaches as in (Minh et al. 2017; Zeng, Gu, and Yao 2018; Hoque et al. 2017; Congjie Wang et al. 2017). A further and probably the most important characteristic of the GA method is that it does not assume any characteristics of the system, but rather considers it a black box.

Taking all the positive of GA it is in its essence designed to solve optimisation problems that have a defined hill or valley it can climb and works well with similar methods. Due to the odd nature of the presented problem the core GA components need to be modified similarly as they are in (Bhondekar et al. 2009). The characteristics of the problem when modifying the method needs to be considered as traditional ways of doing crossing and mutation might affect the system too much as rendering these as useful as the random population generation.

ALGORITHM 1: Modified Genetic Algorithm

```
1  Set popSize ← 500; genMax ← 1000; pop ← genPop(popSize);
2  while genCount ≤ genMax do
3  |  sortPop ← sort(pop , Cost(G,A));
4  |  newPop ← sortPop[1 ... popSize*0.2];
5  |  newPop ← newPop + genPop(popSize*0.3);
6  |  for [indi1,indi2] in sortPop[1 ... popSize*0.6] do
7  |  |  for i = 0 to size(indi1) do
8  |  |  |  newIndi[i] ← randomSelect(indi1[i],indi2[i]);
9  |  |  newPop ← newPop + newIndi;
10 |  for indi in pop do
11 |  |  if random() ≤ 0.2 then
12 |  |  |  indi[random(1,len(indi))]← rand(1,gwCount);
13 |  |  |  newPop ← newPop + indi;
14 |  pop ← newPop
```

Our version of GA can be seen in Algorithm 1 where it takes an initial population size of N for which it generates a new random population of size N. After this, for a predetermined number of iterations or until there has been no change in the utility for many generations the algorithms continue the internal loop. In this loop, individuals with the highest utility that meet the validation criteria are first selected and if not enough are found the remaining spaces are filled with individuals with high utility but who fail to meet the validation requirements. The size of this group is predetermined as a fraction of the total population and is discusses in a later subsection. The next step is the crossing given a certain change of some of the best individuals after which the mutation of some of the best individuals with another varying

chance is performed. Finally, a set of new random individuals is added to the population so local minimum points that might have been missed could be found. This new population is then forwarded to the new iteration.

The GA used for deploying clustered applications onto their respective gateways works in the same way as the global version with certain parameter changes and the validation of solutions being modified. A final difference is linked to the utility function they optimise which does not represent the realistic utility function but just a derivative of this containing all the known and controllable parameters of the apps.

An individual of a population is an array of integers where the indices represent the application id A_i and the value at each point represents the gateway id G_j to which the application is deployed as seen in Fig. 5.2.

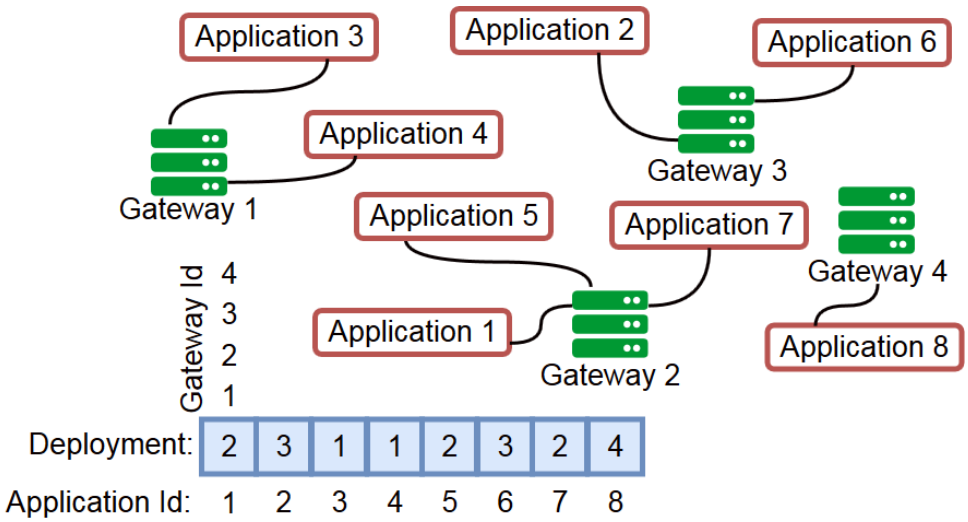

Fig. 5.2 Chromosome used for GA

Generating a new random population is done by creating several individuals for that population. This is done by randomly assigning a gateway to each individual app from the set of gateways, this is done by shuffling a set of the available gateway and allocating the first one to the app.

The elitism component takes the top individuals from a generation and saves them for future generations. In this case, this is done by evaluating the resulting utility if a generation is deployed but also verifying whether a deployment is valid. The elitism has two components, where the first one looks at those individuals that are valid and have the best utilities while the second component considers that if there were not enough individuals that satisfied the validation process then those individuals need to be considered that have a good utility but might not be valid as a lower utility might mean a move towards validating the deployment as

well. This step is mostly done for very large scale and requirements based deployments where a case might arise so that thousands of generations still failed to produce valid individuals.

The crossing of individuals in the method is done by first getting the top N individuals from the generation after which a crossover individual is computed, that has a chance of either having a chromosome from one individual or the other. This method can be extended to cross certain regions of these applications and pick sets but for this implementation as which applications fit together isn't known this would limit the generality of the method. The crossing can be seen in Fig. 5.3 where how these might be picked is shown.

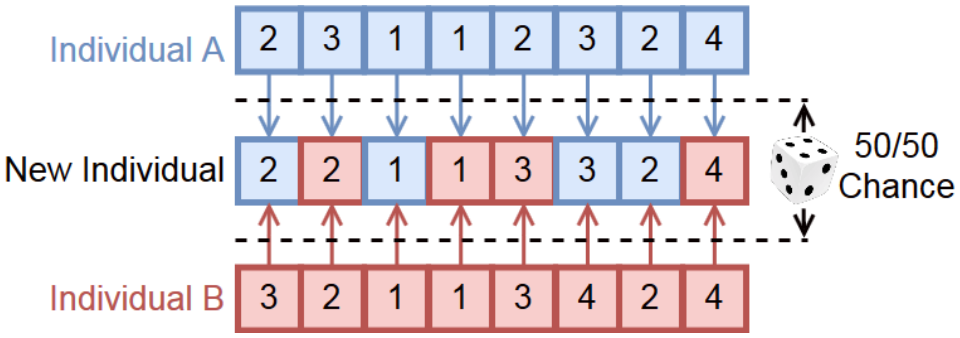

Fig. 5.3 Crossing used for GA

Mutation works in a similar way to crossing as a set of N elite individuals are selected from the current generation go through the chromosomes of the individual and using a random number generator and *mutationChance* constraint the gateway of the application is deployed to is either changed or the old one is kept. This can be seen in Fig. 5.4.

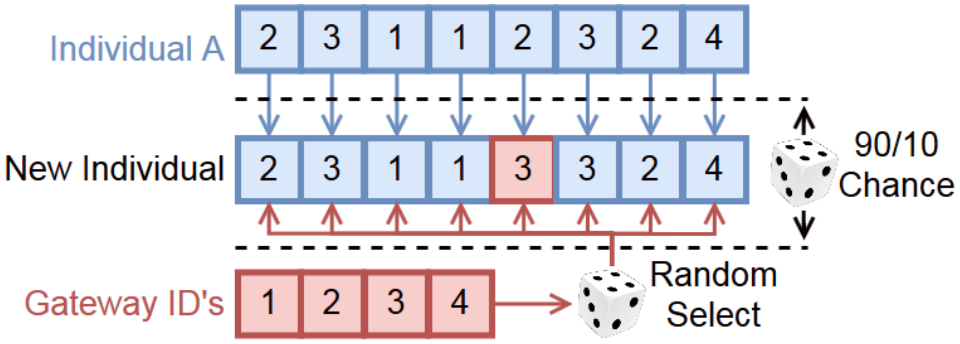

Fig. 5.4 Mutation used for GA

When doing testing, two stopping conditions are typically used. The first and most basic condition is the number of generations condition, which is the most common GA condition where the algorithm is allowed to run for several generations. An extra condition can be added to this where it is allowed to run over this amount if no valid individuals have been

found yet. The second stopping condition and the one used in the tests focuses on stagnation. It looks at whether the GA has found any improvements to the total utility within a predefined number of generations. If no valid individuals were found this, as in the previous case is neglected. Both these cases neglect their stopping conditions if no valid individuals were found. To make sure these methods terminate, safety stops are introduced at very large generation counts.

Considering the Time complexity of such an algorithm a number of parameters influence it, such as the number of generations G , population size P , and chromosome length L . Another component that is considered that other methods don't is the Chromosome Height. Where the length is the number of applications and the height is the number of gateways. These change separately and the gateway changes at about $1/3$ of the rate of the app. All of these determine the total run-time of the System. Taking these into account Random generation, mutation, crossing has a time complexity of $O(P \cdot L)$ or $O(n^2)$ while getting a set of elite individuals has a time complexity of $O(P^2 \cdot L)$ or $O(n^3)$ as the size of the heap to which each individual is compared to needs to be gone through to be able to select the best. This is in line with the work in (Nopiah et al. 2010) where they found that some traditional GA algorithms had a Time Complexity of $O(n^3)$ to $O(n^5)$. Considering that all of these methods only work on a fraction of the population except the elitism, a conservative Time Complexity of $O(n^3)$ can be deducted.

This modified version of the GA algorithm and the subsequent version of it that is used for deployment of clustered applications to their allocated gateways is a good benchmark due to its generic nature and black-box approach to solving the problem. All methods are assessed compared to this one on how well and in how much time they solve certain problems.

5.6 Clustering

The main purpose of clustering is to reduce the computational time of the GA algorithm while trying to find an acceptable solution to the problem. This is done by reducing not just the length but the height of the chromosomes inside a generation. The main challenge with clustering is finding the inflexion zone where the maximum integrity of the system is retained while a large enough speedup of the code is achieved so that its deployment and run-time are feasible or desirable. Considering two disconnected graphs and their individual deployment and thus resulting utility function would have a reduced processing time but still retain most of the original systems integrity. Due to the highly connected nature of the problem, certain connections need to be severed in order to make these disconnected graphs. Minimising

the number of relevant severed connections and creating relevant clusters is one of the key components of the proposed methods.

The four types of clustering methods are increasing in complexity and processing time, but except the random one they are based on DBSCAN or Density based Scan algorithm that is one of the most common algorithms used is Graph analysis for identifying clusters. The presented methods vary in not just their complexity but also on the number of assumptions that need to be made in order to use them.

The Time Complexity formula from GA (Nopiah et al. 2010) that provides the $O(n^3)$ complexity can be modified to look at how the chromosome length and size change when the large deployment is divided into clusters. With a varying cluster size, a reduced complexity of $O(n^2 \cdot log(n))$ is desired. When this is repeated for a number of clusters but whose size is smaller than the chromosome length, an average run-count of $log(n)$ can be considered, resulting in a total approximation of the Time Complexity of $O(n^2 \cdot log(n)^2)$. This results in a large improvement of the optimisation time, especially at larger sizes. This is a very good reason to consider clustering even with the disadvantages and search space reduction that comes with this. This advantage is maintained even if the added processing time caused by the Clustering and Allocation methods is considered.

5.6.1 Random Clustering

Random clustering is the fastest clustering method, but it is also the one that does not attempt to place peers close to each other or close to their resources and endpoints. It is designed as a benchmark for the subsequent optimisation methods and is also used as an initialising agent for the training algorithm.

The algorithm for random clustering is a special case of the Noise Sort algorithm in Algorithm 2. where the distance function from all noise nodes to the clusters is 0 and in this case, the initial cluster size is 0 as well. Here, a maximum value is needed for the cluster sizes and the method will take applications at random and fill up every cluster to its maximum size while attempting to make no clusters smaller than the min size, redistributing these at the end. This method has a very low time complexity of just $O(n \cdot log(n))$? where n is the number of applications as it only needs to go through the applications once and then partially through the clusters. It is the fastest method but the one that in the case of hard requirements and meaningful distances between apps might result in a very bad solution or one that does not have a valid deployment.

ALGORITHM 2: Noise Sort

```
1  Input Apps[],Clusters[];maxPts
2  for i = 0 to size(Apps) do
3      minDist ← MAX_FLOAT; minCls ← 0; alloc ← false ;
4      for j = 1 to size(Clusters) do
5          if size(Clusters[j]) ≤ maxPts then
6              tmpDist ← CalcDist(Clusters[j],Apps[i]);
7              minDist ← min(minDist,tmpDist);
8              minCls ← j;
9      if minDist ≤ MAX_FLOAT then
10         Clusters[minCls].add(Apps[i]);
11     else
12         for j = 1 to size(Clusters) do
13             if size(Clusters[j]) ≤ maxPts then
14                 Clusters[j].add(Apps[i]);
15                 alloc ← true;
16                 break;
17         if !alloc then
18             newId = Clusters.CreateNewCluster();
19             Clusters[newId].add(Apps[i]);
```

5.6.2 Distance based Clustering

The Distance based Clustering works by considering that applications behave in a similar fashion as people do and utilises DBSCAN from Social Network Analysis where it looks at connections between peers and considers that these connections show an interest of working together and deploying them close to each other would improve on the system. In this case, only the app to app connections are considered.

The modified DBSCAN method can be seen in Algorithm 3 where the method is given a *minPts* value which is the minimum cluster size and an *eps* value for which is a minimum distance at which another application is considered a peer, which for 1, means that an application is directly connected with this one, while for two they would have an application connecting them. These connections are directional by nature but for this method, they are considered unidirectional. The method works by going through all the points, checking if they were ever visited and if not checking how many neighbours they have, and if they have more neighbours than the minimum value they form a cluster. This cluster then goes through an expansion phase where all the nodes of the cluster look for additional numbers of neighbours with a size larger or equal to the *minPts* value. If such neighbours are found they are added to the cluster, and this continues until no new points are found. These points in the

ALGORITHM 3: Weighted Clustering

```
1  Input Apps[],minPts, eps ;
2  Set visited ← Array[]; noise ← Array[]; clusters ← Array[];
3  for i = 0 to size(Apps) do
4      if !visited.contains(i) then
5          visited.add(i);
6          neighbours ← getNeighbours(Apps[i],eps,minPts);
7          if neighbours.size() ≤ minPts then
8              noise.add(i);
9          else
10             points ← expandCluster(neighbours,Apps,eps,minPts);
11             clusters.add(points);
12 noiseSort(clusters,noise,Apps);
13 for i = 0 to size(Clusters) do
14     if Clusters[i].size() < minPts then
15         points = Cluster[i] ;
16         delete(Cluster[i]);
17         noise.add(Points) ;
18 noiseSort(clusters,noise,Apps);
```

cluster are then subsequently added to the visited points and are not verified later. If a point fails to find enough neighbours or is not part of any cluster it is considered noise.

When considering clustering, for social media analysis and other scenarios noise is a normal thing and those points are just discarded. This is not acceptable for the application deployment case, so the next step looks at removing those clusters who had enough neighbours but whose neighbours belonged to other clusters. These are removed and added as noise. This noise then needs to be allocated to certain clusters. This is done by adopting methods from the OPTICS (Ankerst et al. 1999) algorithm where the average distance between an app and the connected clusters are calculated and the app is allocated to the nearest cluster or if none is found then its allocated to one at random. This method removes the need for a *maxPts* constraint to be added as the clusters are created based on their peers and these connections need to be retained.

5.6.3 Weights and Attributes based Clustering

Weights and attributes based clustering works in a similar way as the connection based clustering described above, with the difference than rather just looking at one characteristic of applications it looks at all of them. For this model, the parameters of interest were defined as the Distance, Resource Share Rate, Constraints Similarities, Message Rate Similarities,

Unit Load Similarities, Utility Weights Similarities and Requirement Similarities. The selection of the weights can be done manually in the case of a Gray-box scenario where the details of the system are known or the subsequent training algorithms can be used to try and find these values.

When evaluating the parameters that are considered for the custom clustering function the existing app and gateway parameters need to be modified as shown in the equations from Eq. (5.3..5.9). These equations show how the parameters are calculated between apps a1 and a2. These parameters are summed up in Eq. 5.10 where the weights fall into the types $Type_{Cls} \in \{Dist, Share, Constr, MsgRate, ULoad, UtilW, ReqSim\}$.

$$Cls_{Dist} = \frac{1}{ConnDist(a_1, a_2)} \tag{5.3}$$

$$Cls_{Share} = \frac{\sum Resources_{a_1} \in Gateways_{a_2}}{\sum Resources_{a_1}} \frac{\sum Resources_{a_2} \in Gateways_{a_1}}{\sum Resources_{a_2}} \tag{5.4}$$

$$Cls_{Constr} = 1 - \sum^{i \in ConstraintTypes} |Constraint^i_{a_1} - Constraint^i_{a_2}| \tag{5.5}$$

$$Cls_{MsgRate} = 1 - |MessageRate_{a_1} - MessageRate_{a_2}| \tag{5.6}$$

$$Cls_{ULoad} = 1 - |UnitLoad_{a_1} - UnitLoad_{a_2}| \tag{5.7}$$

$$Cls_{UtilW} = 1 - \sum^{i \in UtilityWeights} |UtilWeights^i_{a_1} - UtilWeights^i_{a_2}| \tag{5.8}$$

$$Cls_{ReqSim} = \frac{\sum Require_{a_1} \in Require_{a_2}}{\sum Require_{a_1}} \frac{\sum Require_{a_2} \in Require_{a_1}}{\sum Require_{a_2}} \tag{5.9}$$

$$Cls_{Sum} = \sum^{i \in Types} Cls_i W_i^{Cls} \tag{5.10}$$

Further improvements to the clustering algorithm, especially to its processing time can be done by limiting the scope to which applications look for neighbours. In the base case, they look for neighbours in the whole application set which might be hundreds of individuals. A slight modification would be to consider neighbours that are at a certain distance from the application. If the applications that are considered for the neighbourhood test are limited to be at a connection distance of 4 the total run-time of the method is reduced significantly without affecting the results.

As a conclusion, Clustering methods can greatly reduce the processing time of optimisation methods through the reduction of the search space but if the aims is to have a meaningful reduction more resource intensive methods need to be applied that in some cases can outweigh the benefits of clustering but in others might give better results than the GA.

5.6.4 Eps Value Estimation and Improvements

Having to specify a starting *eps* value for DBSCAN is one of its disadvantages, one that OPTICS solves, but at a great cost as the OPTICS algorithm takes a lot more time to run than the DBSCAN one which at large scales might cause problems. To try and solve this without employing OPTICS a histogram of the distances between apps is generated from which the minimum and maximum *eps* values and the desired steps can be deducted. This method is described in Algorithm 4.

ALGORITHM 4: Eps Estimation and Result Validation

```
1  Input parameterWeights; Apps ;
2  Set distances ← new Array[]; histogram ← new Array[10];
3  for i = 0 to size(Apps) do
4  |   for j = 0 to size(Apps) do
5  |   |   if i != j then
6  |   |   |   distance.add(getDistance(Apps[i],Apps[j],parameterWeights));
7  min ← distance.getMin(); max ← distance.getMax();
8  band = ← (max-min)/10;
9  for i = 0 to size(distance) do
10 |   location = (distance[i]-min) mod band;
11 |   histogram[location] ← histogram[location]+1;
12 iStart← histogram.getMax();
13 for i = 0 to size(histogram) do
14 |   if i>iStart and histogram[i]>Apps.size() then
15 |   |   iStop ← i;
16 return [iStart*band,iStop*band,(iStop-iStart)*band/10]
```

To evaluate the results from each iteration the quality of clustering algorithms outputs needs to be evaluated. Here, there are two extremes, either almost all the applications went into a single cluster or a very large number of applications were generated where a good portion of which have their own cluster. To account for this the method looks at the distribution of clusters which should be as even as possible without too large clusters and without too small ones. The iteration with the best distribution is retained. The algorithm has two stopping conditions, one is if the iteration hit the maximum value and the other is if two

results have been on the extremes after a viable solution was found. Extreme solutions are not considered as viable, because they either make resource allocation impossible or they subvert to the global GA.

These improvements increase the processing time for the algorithm but also make it more generic and results in fewer parameters that need tuning. Variations of the method can be used on scenarios and data-sets that are known to improve the run-time while maintaining the generality.

5.7 Resource Allocation

Resource Allocation focuses on allocating Gateways or shares of a Gateway to clusters in such a way that the upcoming Cluster based GA will produce not just a viable solution, but a solution that improves on the existing global utility. In order to do this, a number of things need to be considered. The first is to make the method as fair and even as possible given the situation. This means that clusters should have a very similar Resource Share Rate based on their Cluster Load. This balance is very high in the Random Allocation and it becomes increasingly worse with the more advance methods as the right allocation becomes more of a priority than the even or balanced one.

5.7.1 Random but Fair

The Random but fair Resource allocation, as mentioned above looks at allocating resources to gateways in as even a manner as possible without considering any other parameters while maintaining gateway integrity as much as possible. This allocation method can be considered a component of the upcoming methods, or more precisely, this method is used to allocate the noise gateways that are left after the initial set of allocations which for the Distance and Resource Share method would be the Cloud Gateways and for further methods those in which no app and subsequently no cluster is interested. The simple algorithm for this method can be seen as a version of the Noise Distribution Algorithm 5 where the clusters apps have no affinity to any gateway. This is the same situation as having the weights set to 0.

The method works by going through each gateway and verifying if all its processing resources have been allocated and if not a cluster with the lowest share rate is found and the gateway is allocated to that cluster in full. Variations of this method could allow gateways to be shared among clusters for an even fairer distribution, but due to the fragmentation of the gateways, the solutions might not be ideal.

ALGORITHM 5: Gateway to Cluster Allocation Noise Distribution

```
1  Input Gateways[]; Clusters[]
2  for i = 0 to size(Gateways) do
3      if Gatewaysi].getFreeLoad() > 1 then
4          if Gatewaysi].getFreeLoad() < 25 and Gateway[i].ClustrCnt() ≥ 1 then
5              Gateways[i].expandClusterShares();
6          else
7              minShare ← MAX_FLOAT; minId ← 0 ;
8              for j = 0 to size(Clusters) do
9                  tmpVal = Clusters[j].getShareRate();
10                 if tmpVal < minShare then
11                     minShare ← tmpVal;minId ← j
12             Gateways[i].allocateFreeResource(Clusters[minId]);
```

5.7.2 Shared Resource Based Allocation

The shared resource based allocation looks at the resources that are linked to a gateway and attempts to allocate gateways to clusters that have the most resources on that gateway. This method also considers that the resource share of the gateways is important as the resulting share is divided by the resource share if the allocates share is larger than 120% of the allocated load. The share values are multiplied by a factor of 10 when the minimum share rate of 120% of the minimum required is not fulfilled. This gives clusters that haven't fulfilled the minimum requirements an edge over those that have. Also while there are clusters that haven't met the minimum share rate other clusters are not allowed to demand resources.

5.7.3 Weighted Property based Resource Allocation

To allocate gateways to clusters a similar distance needs to be defined as in the application clustering section. In this case, the Gateways distance to the Clusters or more precisely to the Application of the clusters is central. In the random allocation, this component is not considered and in the case of Resource share allocation, only the resource is considered.

When considering the parameters that are need to show the preference of an application or a cluster of applications to a gateway the ratio of these parameters needs to be modified as shown in Eq. (5.11..5.15). These equations show how the parameters are calculated between app a1 and gateway g2. These parameters are summed up in Eq. 5.16 where the weights fall into the types $Type_{Alloc} \in \{ResShare, BaseToULoad, PerfCapToULoad, SpeedToULoad, Capab\}$. How this is calculated for Clusters is shown in Eq. 5.17

$$Alloc_{ResShare} = \frac{\sum Resources_{a_1} \in Gateways_2}{\sum Resources_{a_1}} \tag{5.11}$$

$$Alloc_{BaseToULoad} = 1 - |\frac{L_B^2}{100.0} - \frac{L_u^1 - Min_{L_u}}{Max_{L_u} - Min_{L_u}}| \tag{5.12}$$

$$Alloc_{PerfCapToULoad} = 1 - |\frac{P_j^{Cap} - Min_{PCap}}{Max_{PCap} - Min_{PCap}} - \frac{L_u^1 - Min_{L_u}}{Max_{L_u} - Min_{L_u}}| \tag{5.13}$$

$$Alloc_{SpeedToULoad} = 1 - |\frac{P_j^{Speed} - Min_{PSpeed}}{Max_{PSpeed} - Min_{PSpeed}} - \frac{L_u^1 - Min_{L_u}}{Max_{L_u} - Min_{L_u}}| \tag{5.14}$$

$$Alloc_{Capab} = \frac{\sum Require_{a_1} \in Require_{Gw_2}}{\sum Require_{a_1}} \tag{5.15}$$

$$Alloc_{Sum} = \sum^{i \in Types} Alloc_i W_i^{Alloc} \tag{5.16}$$

$$Alloc_{Clust}^{i} = \sum^{i \in Types} Alloc_i W_i^{Alloc} \tag{5.17}$$

The Algorithm 6 works by first creating a list of all the gateways each cluster is interested in with their specific share rate or distance function. After this is compiled the algorithm loops until all clusters run out of gateways of interest and the noise sorting or random but fair allocation is done. In the loop, there are several special cases considered. First, if a cluster has received 90% of the average share rate of the System and more than 120%, all the gateways are removed from its preference list.

The main body or the loop of the algorithm goes through all the valid clusters, which then select the gateway that they are most interested in. All the other clusters that are interested in this gateway then compete for a share. Only clusters that have their distance or share rate within a fraction given as a constant are considered. The number of clusters that can share a gateway is also limited by a fraction constant that is given, so even if some gateways are within the threshold if they are outside that constant they are not considered. After a gateway is allocated it is removed from all clusters that are interested. If the selected gateway falls outside the threshold, it is not considered.

ALGORITHM 6: Weighted Property based Resource Allocation

```
1  Input Gateways[]; Clusters[]; globalShareRate; clsShareRate;
2  Set EmptyClusters ← new Array[]; ClsInterrest ← new Array[];
3  allocatedGWs ← new Array[]; for i = 0 to size(Clusters) do
4      for j = 0 to size(Gateways) do
5          if distance(Clusters[i],Gateways[j]) != 0 then
6              ClsInterrest[i].add(Gateways[j],distance(Clusters[i],Gateways[j]));
7  while Clusters.size() < EmptyClusters.size() do
8      for i = 0 to size(Clusters) do
9          if Clusters[i].getShareRate()>0.8*globalShareRate or
             ClsInterrest[i].size()==0 then
10             EmptyClusters.add(Clusters[i]);
11         else
12             maxGwId ← ClsInterrest[i].getMaxGw() ;
13             maxDist ←ClsInterrest[i].getMaxVal() ;
14             competeCls ← new Array[];
15             for k = 0 to size(ClsInterrest) do
16                 if k!=i and ClsInterrest[k].contains(maxGwId) then
17                     competeCls.add(k);
18             topCls ← competeCls.getBestCls(clsShareRate);
19             if Clusters[i] not in topCls then
20                 clsInterest[i].remove(maxGwId);
21             else
22                 clsInterest.removeAll(maxGwId);
23                 distributeGwToClusters(topCls,maxGwId);
24 GatewayToClusterNoiseSort();
```

5.7.4 Correlation and Weights based Resource allocation

Correlation and Weights based Resource allocation is a variation of Distance based clustering method where the distance between one gateway and a cluster is not just defined by the share rate, but rather a set of variables that are given a certain importance based on their weighting.

The varying parameter based distance supported allocation is crucial to the similar style clustering as they are complementary methods. There is little reason to only consider advanced clustering methods and then just allocate random gateways to these as they would defeat the core purpose of these methods which is to put the application to their most preferred gateways within the best conditions possible in a sensible amount of time. It is also worth noting that if the applications in a cluster are allocated randomly the clusters preference to gateways will be week, so the algorithm will not be able to perform as well as it could with proper application clustering.

This version of the resource allocation proposes an initial allocation set for the clusters. These allocations could later be revisited by analysing failed Clustering GA results to see which component or parameter of the allocation failed and increasing the weights of those parameters and having two or more clusters re-negotiate their gateways with an incentive to reach an agreement. These fall under the iterative methods that can be used to improve tweak the algorithm.

The resource allocation methods provide the backbone for the system as they allocate the gateways to the clusters which could very well be random on initial deployments or in the case of poorly selected weights. Besides the distance based allocation, the main task of this method is to make sure that not just the best possible result is found but allocations are made that produce viable deployments as the whole deployment direction is considered a failure if just one cluster fails.

5.8 Overview of Methods

5.8.1 Connections based Clustering and Resource Allocation

Network Analysis Clustering and Allocation in Fig. 5.1 route b. is designed to support the work that was done in (Verba, Chao, A. James, Lewandowski, et al. 2017) where a number of use case scenarios were analysed with a varied number of Graph analysis methods and an extensive characteristics analysis was conducted to analyse their choice of peers and gateways as well. Through this research, it has become obvious that when considering clustering and deployment for the presented Delay optimisation Scenario the parameters that most commonly linked application together were their connection. Their choice of gateways was determined by whether their resources were deployed on that gateway or by the gateways latency to those resources.

Due to the method sole interest in the connection of application for clustering the method used to find neighbours and clusters is a lot less resource intensive than the parameter and characteristics ones as this one can easily be solved by a Dijkstra algorithm and as such is easily distributable and scalable. This makes it the best method for optimising typical scenarios. The sorting of the noise applications that do not fit into any clusters can also be done more easily as they could be considered individually and ping clusters for their parameters and then join these when preferred.

The resource allocation based on shared resources poses the same advantages as with the clustering as finding gateways that host the application's resources is a lot less processing time intensive than finding ones that have desirable parameters/characteristics. This also

allows this method to be distributable and is thus better scalable. The allocation of Cloud Gateways and those that have no resources allocated to them is done in a similar fashion as with the clustering with these being allocated by preference or to the gateway with the lowest share rate.

The final part of the method involves running a Clustering GA method where the applications inside a Cluster are deployed onto the gateways allocated to this cluster while making sure the deployments are valid. This is the final step in the solution and it shows how well the method worked. Compounding the best individuals from the clustered deployments results in the global deployment and finally the complete and global utility function.

5.8.2 Iterative Correlation based Clustering and Optimisation

When considering more advanced or difficult deployments such as the ones presented in (Zeng, Gu, and Yao 2018) and in the utility model in chapter 5 only considering certain parameters when clustering applications and choosing peers as well as allocating resources might be detrimental. In order to be able to apply the clustering methods to a larger scope of models and problems, there needs to be a generic template on how to measure distances between peers and resources and how to allocate these. This approach aims to solve some of the problems inherent with making initial assumptions of the system and proposes methods of deducing these characteristics and then enforcing them when considering Deployments and modifying them when needed. This method is the most processing time intensive one as it has a number of high complexity components, but it is the only one that is generic enough to be used all across this problem and use-case domain.

5.8.2.1 Overview of Method

A general view of how this method works can be seen in Algorithm 7 where the components interaction and work-flow is shown in detail. This method can be expanded to add intermediate tuning components and fault and fail based intermediate iterations which would reduce the computational time of components but it would still fall under the same category of problems and fit in the presented diagram.

The correlation between parameters and deployments is calculated by taking a set of possible best-case deployments and looking at the relation between parameters when they are deployed together and when they are not. In this case, the correlation is calculated between a Boolean value of whether they are deployed together or not and a double value which is the parameter relation value. Due to this method, there is a lot of noise when calculating the correlation, and because of the uncertainty of the quality of the solution, the correlations

ALGORITHM 7: Iterative Correlation based Clustering and Resource Allocation

```
1  Input weights[]; applications[]; gateways[]; corrTrainer;
2  Set stopStatus ← false ;
3  clusters ← new Array[]; bestDeployment ← new Array[]; bestUtil ← 0.0;
4  while !stopStatus do
5      [results,clusters] ← weightedClustering(weights,applications);
6      if results.status == "success" then
7          weightedResourceAllocation(weights,clusters, gateways);
8          [results, deployments[]] ← GADeployment(clusters);
9          if results.status == "success" then
10             [tmpUtil,deplId] = getBestDeplUtil(deployments);
11             if tmpUtil>bestUtil then
12                 bestUtil← tmpUtil; bestDeploymetn ← deplId;
13             corrRes ← calculateCorrVals(getBest(deployments,5));
14             corrTrainer.updateParameters(corrRes,tmpUtil);
15             weights ← corrTrainer.getNewWeights();
16         else
17             corrTrainer.setFailed(results);
18     else
19         corrTrainer.setFailed(results);
```

might be very small or not representative of a good direction at all. The correlation parameters are the ones mentioned in the previous subsections for weighted Clustering and Weighted Resource allocation. The data for the correlation is generated by creating a list of applications for each application that are deployed together, calculating their parameters and adding them to the data set with a Deployed value of 1. After this, data is added with the applications not in the list and a Deployment value of 0. The data is filtered in such a way that only unique data points are added and no duplicate entries such as A_1's relation to A_2 and A_2's relation to A_1.

5.8.2.2 Correlation Calculation

The correlation is calculated using the Pearson R Correlation (Pearson 1895) where $R[x,y]$ is considered as the correlation between parameter x and parameter y. The Pearson R Correlation is considered as opposed to the Spearman's Correlation (Spearman 1910) based on the work done in (Hauke and Kossowski 2011) where it is stated that the results of the Spearman's Correlation can be less reliable with some data sets but might find certain correlations that the Pearson's misses. This is due to the understanding that Pearsons's looks for linear Correlation between variables while Spearman's is looking for a Monotonic one.

Furthermore, Spearman's method requires the Ranking of data which implies knowledge about the data. As a conclusion, the Pearson's correlation is used as it requires less implied knowledge of the data and has a lower chance of offering a false positive for the correlation.

The formula for the correlation can be seen in Eq. 5.18 where n represents the total number of data-points, x and y represent single data points, $\bar{x}$ and $\bar{y}$ are the mean values of these data points while S_x and S_y represent the standard deviation for x and y. The mean of x and y is calculated by adding all the values and dividing them with n. The standard deviation for a parameter is based on Eq. 5.19. This is done by first calculating the sum of all differences squared between each individual point and the mean of that parameter. After this, the sum is divided by $(n-1)$ where n is the total number of data points.

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{S_x\, S_y} \tag{5.18}$$

$$S_x = \sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{5.19}$$

The following calculations are completed for all parameters and in relation to the deployment status which is the property noting whether two application are deployed together. An example of this can be seen in Table. 5.1 for the Delay Optimisation scenario where the resource share and distance measurement parameters have a lot higher correlation than the other ones, even though these values are still under 0.5 resulting in a week correlation at best. The reason for this is partially due to the sub-optimal parameters and the high levels of noise in the data. Here, all the 7 parameters for app to app relations and the 5 parameters between app to gateway relations are shown.

Table 5.1 Example Correlation Results

Clustering Parameters

Dist	Share	Constr	MsgRate	ULoad	UtilW	ReqSim
0.159041	0.277061	-0.005882	-0.021282	-0.002517	0.0	0.004135

Allocation Parameters

ResShare	BaseToULoad	PerfCapToULoad	SpeedToULoad,	Capab
0.382969	-0.006193	-0.042599	4.46E-19	0.0

Taking these correlation values, the algorithm is applied, that looks at the highest correlation and applies a constant processing Limit to see which other parameters are within range of the best ones. After this is done, the selected correlation values are given adjusted based on the predefined penalties for certain parameters and then they are adjusted so the sum of

weights is equal to 1. These parameter specific weights are then used do the Clustering and Resource allocation with the specification that those parameters that are not considered are not calculated as well when clustering or allocating.

5.8.2.3 Weight Tuning Method

The Weight tuning method falls under the category of Costly Global optimisation (CGO) methods. This method is considered a CGO because of the costly computation that goes into verifying the validity of a tuning step through the clustering, allocation and actual deployment of the applications based on these. This method works by controlling the processing limits that are considered when calculating the weights and by adjusting penalties for certain parameters. This is done through a number of steps and taking into account the location of certain fails and the cause of these. These steps are Probing, Underfitting, Overfitting and Undefined Stagnation Resolution.

ALGORITHM 8: Weight Tuning

```
1  Input correlationResults[]; ClusteringStatus; DeploymentStatus;
     previousWeights[]; maxStep; failCnt; utility; prevUtility; utilDiffLim;
2  Set exit ← false; adjustments ← new Array[];
3  if failCnt ≤ maxStep then
4      if utility < prevUtility+utilDiffLim or ClusteringStatus=="Failed" or
         DeploymentStatus == "Failed" then
5          if uderfitAppCheck(weights) then
6              adjustApp ← underfitCompensationApp();
7          else if overfitCheck(weights) then
8              adjustApp ← overfitCompensationApp();
9          else
10             adjustApp ← randomAdjustmentsApp();
11         if uderfitGwCheck(weights) then
12             adjustGw ← underfitCompensationGw();
13         else if overfitCheck(weights) then
14             adjustGw ← overfitCompensationGw();
15         else
16             adjustGw ← randomAdjustmentsGw();
17         weights ← probeDirection(correlationResults,adjustApp,adjustGw);
18         return weights
19 else
20     return exit ← true;
```

The main body of the algorithm can be seen in Algorithm 8 where an initial set of deployments with the best utility function and a set of correlation parameters are considered.

If no comparison exists or if the new solution is the best so far then the Probe sequence continues with the current direction. If the received solution fails the direction stop criteria then a change needs to be made to the existing setup. In this case, the results and the system are analysed to determine the cause of stagnation or worse solution which could be Underfitting, Overfitting or Undefined. If the received solution fails the full Stop Criteria then the algorithm is terminated and the best viable solution is used.

The direction stop criteria are used to verify whether the current direction of the tuning is a correct one or new directions need to be explored. This condition returns true if it is the first iteration of the method and false if the attempt failed for any reason. Furthermore, the method returns false if there is only one parameter considered and the tests on that results have come in as well as in situations where the weights have not changed significantly enough compared to previous attempts or the improvement of the utility function that was made is too small to warrant further probing.

The full stop criteria is triggered when the direction stop criteria returns a false response and the maximum number of failed attempts from a single point have been reached or the maximum number of total iterations has been reached. The algorithms counts the number of times the method attempted to find a new minimum point from an existing best point. If it finds a new minimum point the fail count is reset.

The probing method is called when all the criteria are satisfied but also when the direction criteria is not and changes have been made through one of the three changing methods. In the second case the probing method would have its constants changes or modified and it would use the best solutions data to find the new weights as the current solution is considered a bad direction. This method works by applying the correlation calculation mentioned above and if new weights are found based on the constraints given a new deployment is attempted.

The Underfitting method is concerned with verifying whether the solution that is attempted is too generic and whether that might be the problem why the system is stagnating or no better solution can be found. It verifies whether this is the case by looking and the compensated values of the weights and whether their mean is below a certain set threshold. If this condition is met then the method will attempt to correct his problem by Increasing the processing Limit by a constant that is intensified or multiplied by the current fail count.

The Overfitting Method looks at verifying if the current solutions are focusing too much on a parameter and if that is the reason for the stagnation or bad results. It first verifies that this is the case by looking at whether any parameter has their weights higher than a given constant. If such a case is found then the processing Limit is decreased by the same constant as before and multiplied by the current fail count. A further step is taken by adding an extra penalty to the parameter in question. The penalty is based on a penalty constant.

The Undefined Change Method is used when the reason for the stagnation is not known or cannot be found. This happens if all previous validations fail. In this case, the processing limit is kept the same. The average weight of properties is calculated and given a negative penalty to the high ones and a positive penalty to the low ones in the first attempt and the other way around in the second attempt.

When and how these methods are used and to which set of weights app to app or app to gateway can be altered based on the results of the deployment as well based on which part failed. Redoing clustering weights if the found clusters were very uneven or if the clustering itself failed is an option. Redoing the allocation weights if the Resource allocation or deployment failed would also speed up the system by not requiring a full analysis or it could be done when a failure occurs.

The iterative correlation method, while allowing the user to forget certain constraints and allow the algorithm to just run on its own has significant disadvantages when it comes to the impact on the processing time as highly complicated and resource intensive algorithms needs to run multiple times to train parameters or evaluate existing ones. Despite this, this method is the only one that is designed to work with an unknown system with varying parameters and objectives and attempt to make assumptions about that system in a combination of probabilistic and deterministic approaches.

5.8.3 Sampled Data based Correlation and Weight Calculation

Considering the high processing time and complexity of the Iterative clustering an correlation calculation, a case could be made that there is no reason to analyse the whole system but rather taking a representative chunk out of the system and analysing the behaviour of those parameters as well as looking at just a small portion of the clusters deployment for later analysis might yield similar results with a major reduction in complexity and processing time. The problematic component is figuring out what a representative sample means, how small or large it has to be.

The creation of the initial sample can be done in several ways. The first and most low cost but risky method is just selecting a fraction of the application that needs to be deployed on the system, and considering them as a cluster, while allocating resources to them based on the weighted allocation method and to point where they match the resource share of the system. This method is seen in 9. The second method of doing this is a more deterministic approach where the general characteristics of the large-scale system are considered and these are replicated in the small-scale deployment, not just the share rate, but application types sizes and connections. If the consequent deployment fails or exceeds the maximum allocated time for the run a small sized cluster is selected until a valid deployment is reached. The

ALGORITHM 9: Sampling Algorithm

```
Input initialWeights[]; apps[];gateways[]; maxFailCnt; reqSize; minClsSize;
Set failCnt ← 0; clsSize ← reqSize;
while failCnt < maxFailCnt do
    cluster ← generateRandomClusterOfSize(apps; clsSize);
    assignGwToCluster(cluster,gateways);
    [results,deployments[]] ← GADeployment(cluster);
    if results == "Success" then
        break;
    else
        if clsSize/2<minClsSize then
            clsSize ← minClsSize; failCnt ← failCnt + 1;
        else
            clsSize ← clsSize / 2;
failCnt ← 0;
while failCnt < maxFailCnt or size>=reqSize do
    corrRes ← calculateCorrVals(getBest(deployments));
    weights ← getNewWeights(corrRes,adjustParams);
    cluster ← createCluster(apps;clsSize;weights);
    assignGwToCluster(cluster,gateways,weights);
    [results,deployments[]] ← GADeployment(cluster);
    if results == "Success" then
        clsSize ← clsSize+clsSize/(2+failCnt); failCnt ← 0;
    else
        failCnt ← failCnt + 1;
        adjustParams ← modifyAdjustmentParameters(weights,failCnt);
return corrRes ← calculateCorrVals(getBest(deployments));
```

worst case scenario is that of one application being allocated to one gateway in a valid way. Larger cluster sizes are then attempted until a successful cluster deployment is found that is of an appropriate size.

Using this cluster and deployment the GA is run with a now higher termination count, generation size and population count which would result in good deployment solutions for the set. Based on this set, the *eps* value is calculated and scale it based on the tests conducted in (Verba, Chao, A. James, Lewandowski, et al. 2017) to account for the increase in connections with the increase in scale.

In the subsequent iterations, a sampling of the resulting clusters is considered as a reference for further deployment and analysis. The number of clusters is determined by the selected fraction of sampling size which then determines the accuracy of the evaluation and its results. This can also be done in two ways, the first being a random selection of clusters

another being a deterministic selection where the ones are selected that have the highest individual application utility while retaining System characteristics and resource share rates. After these clusters are selected they are combined and their results are analysed as with the global method presented in the previous chapter.

Due to the sampling nature of this approach, there is no guarantee that the selected individuals behave characteristically to the whole system. This method may also make the impact of positive feedback loops more prominent and have a wider effect on the system. On the other hand, because this system allows for solutions to scale back to sizes where finding viable deployment becomes feasible and then scale up again, this approach might have a higher chance of finding solutions in a difficult highly constrained setting, as the previous method requires a viable solution to arise from the first initial deployments.

This method attempts to solve some of the complexity and scalability issues with the previous one by sacrificing some of the generality that comes with it. It is also an attempt to provide a solution for instances where finding any viable individual or deployment becomes a difficult task in itself.

5.9 Summary

When considering the presented methods, they have a varying level of implementation complexity, processing run-time and generality. Methods like the Random Clustering and Allocation, Distance-based clustering Share-based Allocation and sampling based Iterative optimisation are distributable with slight modifications so in theory could be used to optimise systems with thousands of applications and devices. For such systems, it is becoming apparent that a higher level of separation might be needed where these systems might need to be split up to different Fogs as their high level of heterogeneity and varying utility requirements might create systems for which no single distance function or allocation method may apply. Here, the creation of smaller Fog deployments is needed, where their utilities and characteristics are similar enough so that their deployment warrants more advanced methods as the ones described in this section.

The Processing costs of the methods and their generality are compared in Fig. 5.5. From these, it is obvious, that in order to gain a certain sense of generality added processing power needs to be allocated. Here, while the Global GA is the most general method it is still the most resource intensive and while the random allocation is the simplest to implement and fastest it may only work in certain cases and might cause undeployable allocations when Requirements or hard constraints are introduced into the system.

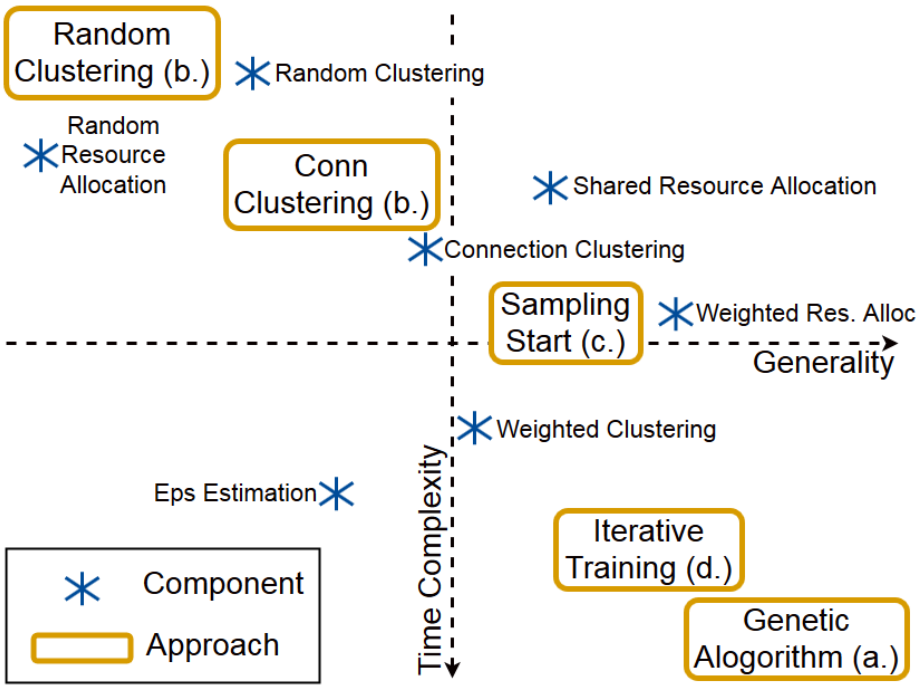

Fig. 5.5 Overview of Methods

In this chapter, clustering methods and one generic method were presented that aim to solve the QAP of allocating interconnected applications to gateways and maximising their utility as well as the system utility. A number of clustering methods have been explored with various capabilities and complexities. These methods will be evaluated in the upcoming section based on the specified use cases where the differences in applicability and processing requirements will be shown through rigorous testing and comparisons on scalability tests.

Chapter 6

Evaluation and Analysis

The evaluation and analysis chapter follows three distinct evaluation processes. These are in line with the components presented in the methodology section. The tests are designed to accomplish two things. First, a set of tests are conducted to determine the testing scenario, generations, model and testing parameters. Secondly, these tests aim to evaluate the proposed methods and their components in such a way that their impact on the scalability, quality of the outcomes and execution time can be determined.

The first set of subsections is designed to provide an evaluation scenario and to analyse the accuracy of the proposed application and gateway model. This is done by proposing four industry based use-case scenarios from which scalability and interconnectivity parameters are derived using graph analysis tools. After these parameters are found a set of single, bundled and migration deployments are performed to evaluate the proposed model's accuracy.

After the model is evaluated, the testing parameters and objectives of the optimisation methods are fixed. First, the three optimisation scenarios and their utilities are defined and the GA algorithm is evaluated to find the parameters where it performs best. This is done so that the proposed methods quality is evaluated and not the quality of the modified GA.

The subsequent validation tests can be split into three categories. The performance analysis looks at comparing the proposed methods to some standard scenarios or variations to show how the resulting utility and execution time changes with the scale and scenario alterations. This provided an overview of how results are achieved. The scalability tests look at how the proposed methods function on higher scales where the only interest is in the outcomes and execution time, the path towards those is not relevant. These tests are there to show the large-scale affinity of the proposed methods and the advantages of clustering in general.

The component evaluation section is designed to showcase the individual contribution of the method components in both the execution time and the resulting outputs. Here a number

of varying methods that have the same objectives are compared while fixing every other parameter to showcase their capabilities.

6.1 Analysis and Replication: AME Case Study

6.1.1 Use Case Description

The presented use cases are based on the 4 physical workstations available at the Institute For Advanced Manufacturing and Engineering. The proposed automation and control systems are in concurrence with the requirements of the industrial partner and those presented in Industry 4.0.

6.1.1.1 Physical Systems

The Dimension Testing Metrology station contains a Coordinate Measuring Machine (CMM), alongside some smaller measurement devices, and an environment monitoring station for accurate temperature and humidity control which is essential for accurate measurements as well as a monitoring screen and a parts organising station. This workstation is designed to measure tolerances on finished components as well as bending and torsion. The key factors here are linked to quality assurance, environmental monitoring and Energy Control and monitoring.

The Metallurgy Metrology workstation contains a Hot Mould Machine, a Polishing Controller, Digital Microscopes, a part organiser and monitor. This workstation is used to take weld pieces, mount them into plastic moulds, polish and analyse these for integrity. The key factors here are part monitoring, tests logging and quality control.

The Stress testing workstation contains a Compression testing, Burst testing and Stretch testing instruments as well as parts organiser and monitor. This station is used to test the integrity of welded tubes under pressure through the burst tests, as well as component characteristics through the compression and stretch or pull tests. The key components are regarding parts monitoring and tests logging together with energy monitoring and quality control.

The Assembly line contains several ABB Robot arms with 2D vision capabilities together with welder units, a conveyor belt with position sensors, controls and barcode readers, an input and output part organiser, safety proximity laser curtains and emergency stop buttons.

The assembly line is used to weld and assemble components going through the line based on their part numbers. The key components are part monitoring as well as quality control through the metrology stations, safety and energy monitoring and control.

6.1.1.2 Application Use Cases

The design of the application use-cases are based on the existing hardware and sensor environment as well as guidelines presented in (J. Lee, Bagheri, and Kao 2015). The main purpose of these systems is to map flow based energy control, part monitoring access and environmental control on top of existing hardware with a realistic composite application approach. Each scenario has a different approach to the topology of the connections. The part Logging system is designed to be a more connected design while the energy monitoring and access control scenarios are more hierarchical or resemble fractal and tree based graphs.

6.1.1.3 Part Logging and Flow Monitoring

This system is designed to monitor the progress of parts through the assembly and metrology environments as well as gather data on parts production rate and use per environment as well as receive controls from the energy optimiser on where to assign parts.

The virtual connections of the system can be seen in Fig. 6.1 where each component or application is shown with its respective Cloud, storage, local access and device connections. The graph shows that the applications are highly connected between each other while the devices usually belong to one controller/orchestrator or reader with no direct machine to machine (M2M) communication between devices.

The use-case contains a main parts flow monitor and a status monitor component which is then connected to a local component for each room which communicates with each individual machine type, controller and reader. There is a local repository for parts status monitoring for each workstation as well as local access. Finally, there is a Cloud monitoring connection for saving data and advanced analysis.

6.1.1.4 Energy Monitoring and Control

The system is designed to monitor the energy use of devices and machines for each workstation and the factory as well. It also controls the power supply of machines based on parts flow and existing optimisation scenarios. These parameters are shown on displays and saved to a Cloud source.

The virtual connections of the system can be seen in Fig. 6.2. The diagram shows that the connections in this scenario are much less clustered and more hierarchical than in the previous scenario especially for the left half, which is the control region, while the right is the monitoring and optimisation part.

The presented use case contains a Cloud-connected main power controller linked with local controllers that have local access and that in hand orchestrate the individual devices.

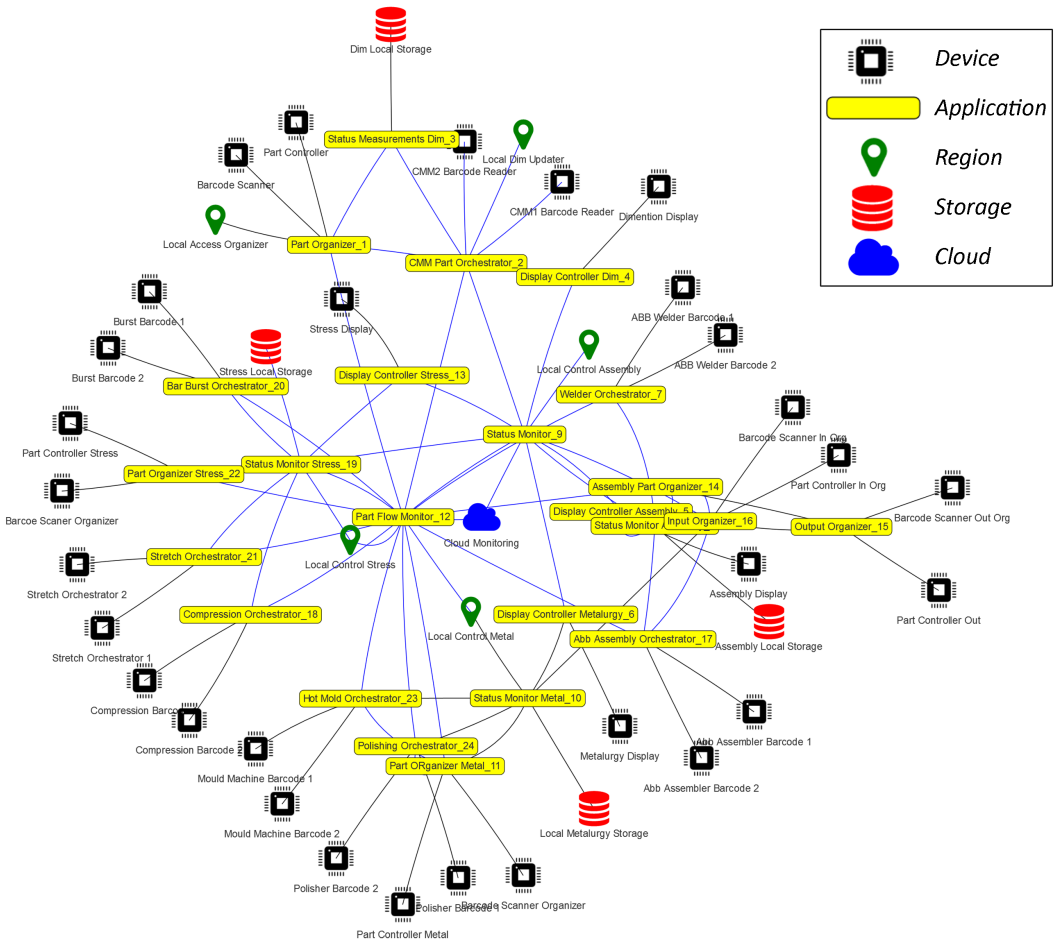

Fig. 6.1 Parts and Flow Monitoring subsystem

This component is linked with the Energy Optimiser which is connected to the flow monitor and Main Energy Monitor.

The main monitor is linked with local monitors that save data to local storage and show info on local displays while saving data for further analysis on a common Cloud Energy Monitor endpoint.

6.1.1.5 Access, Safety and Environment Control

This system is designed to take care of controlling and logging access on machine and rooms as well as controlling safety and environmental variables inside the rooms. Cloud logging, control, access and displays are connected to these components.

The graph of these connections can be seen in Fig. 6.3 where the graph has a similar structure to the one in Fig. 6.2, but containing more local access points and a much more hierarchical system which is designed for layered safety in the case of access and security.

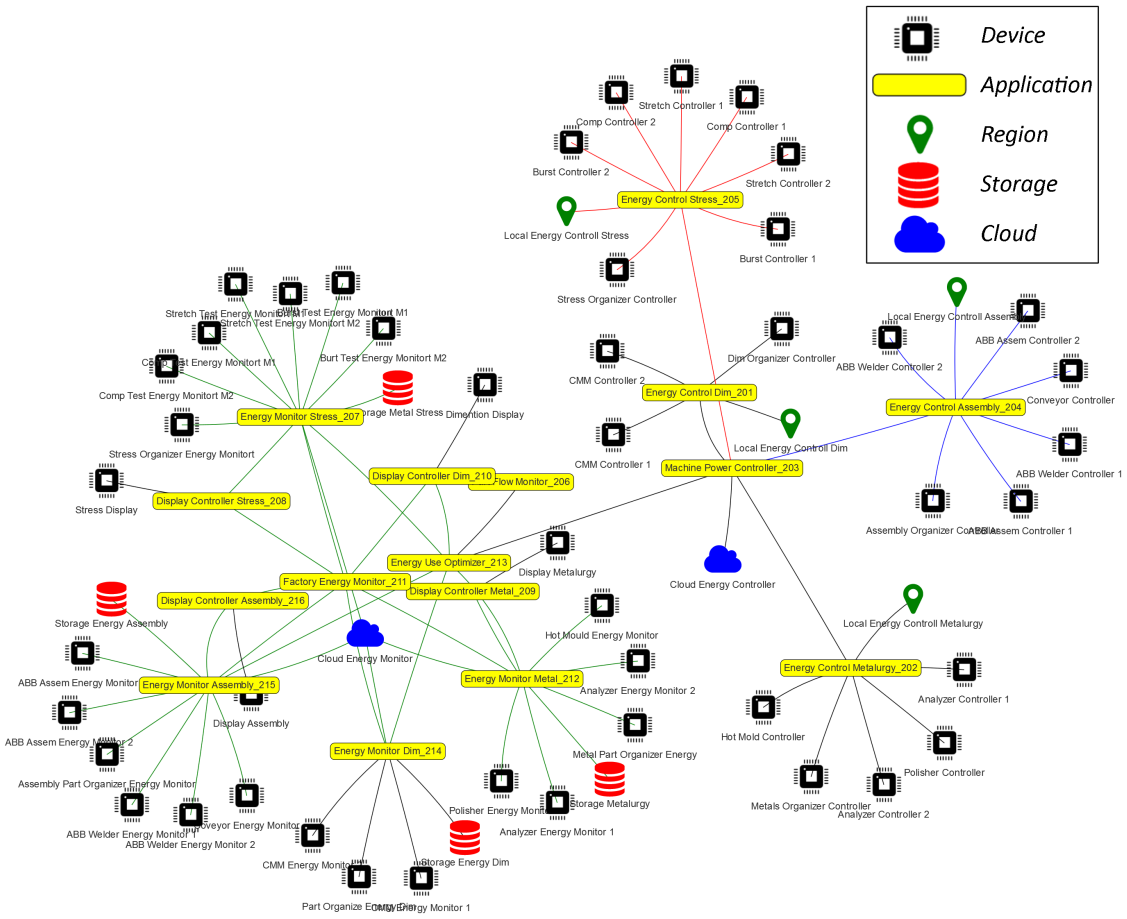

Fig. 6.2 Energy Monitor and Control subsystem

This scenario contains the main access manager that controls the room access, parts access and machine access modules that in hand orchestrate the room modules and their devices. The access manager is linked to the safety controller which in hand is linked to the environmental controller to initiate safety protocols if needed. The safety controller is linked to individual room components that in hand control the safety devices and sensors available. The environmental components orchestrate ventilation, temperature and humidity control through factory level components. It also has specialised units for the high precision environment control requirements of the Dimension measurement workstation which increases the number of sensors and splits the humidity and temperature as well as the ventilation.

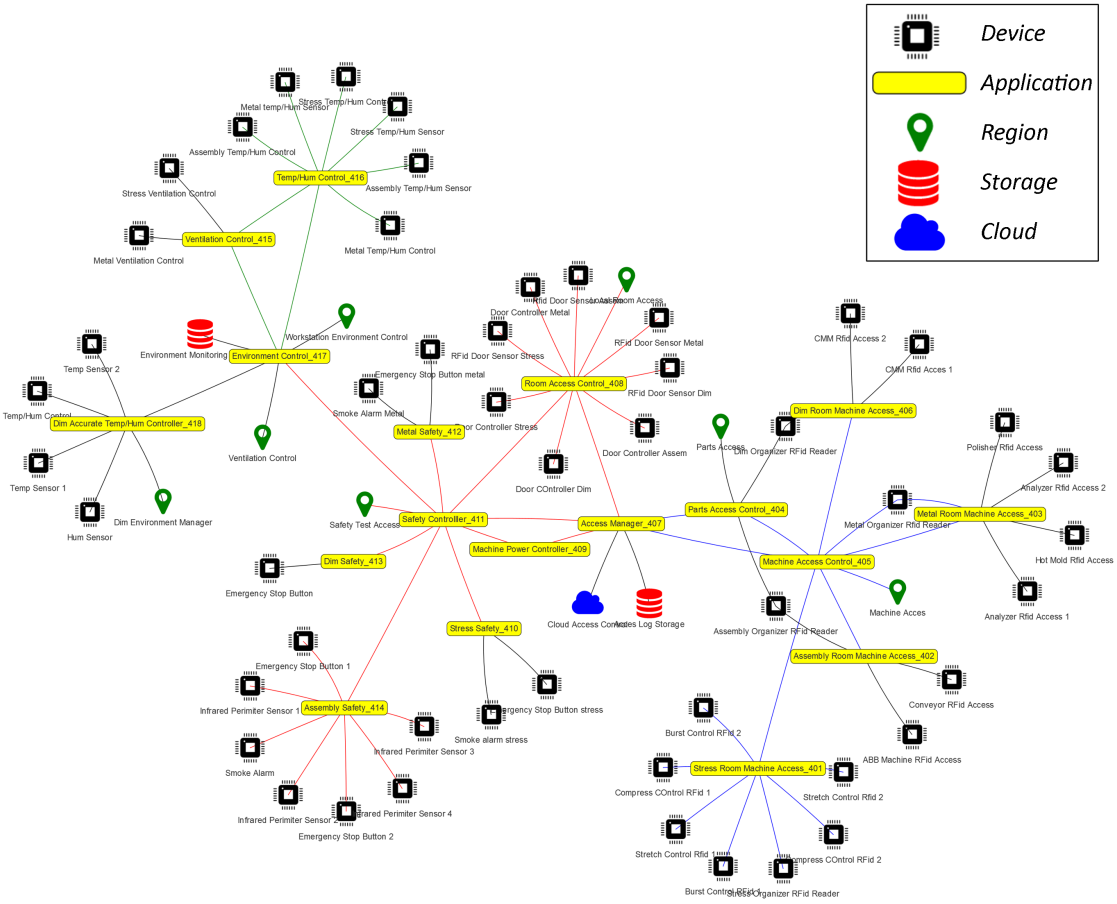

Fig. 6.3 Access, Safety and Environmental Monitoring and Control

6.1.1.6 Combined System

The combined system can be seen in Fig. 6.4 looks at connecting the separate systems for a fully functioning factory floor. This is done by linking certain main components in these systems through a layered architecture design.

The main connected components that are the part flow controller with the energy use optimisation that connects to the machine part controller which then relates to the Safety Controller and the Access Manager.

6.1.2 Analysis Parameters

When considering the analysis of IoT systems, there are several parameters that need to be examined that may be interesting for two reasons. The first reason is for replication and scaling of these systems when testing how optimisation algorithms perform with larger data

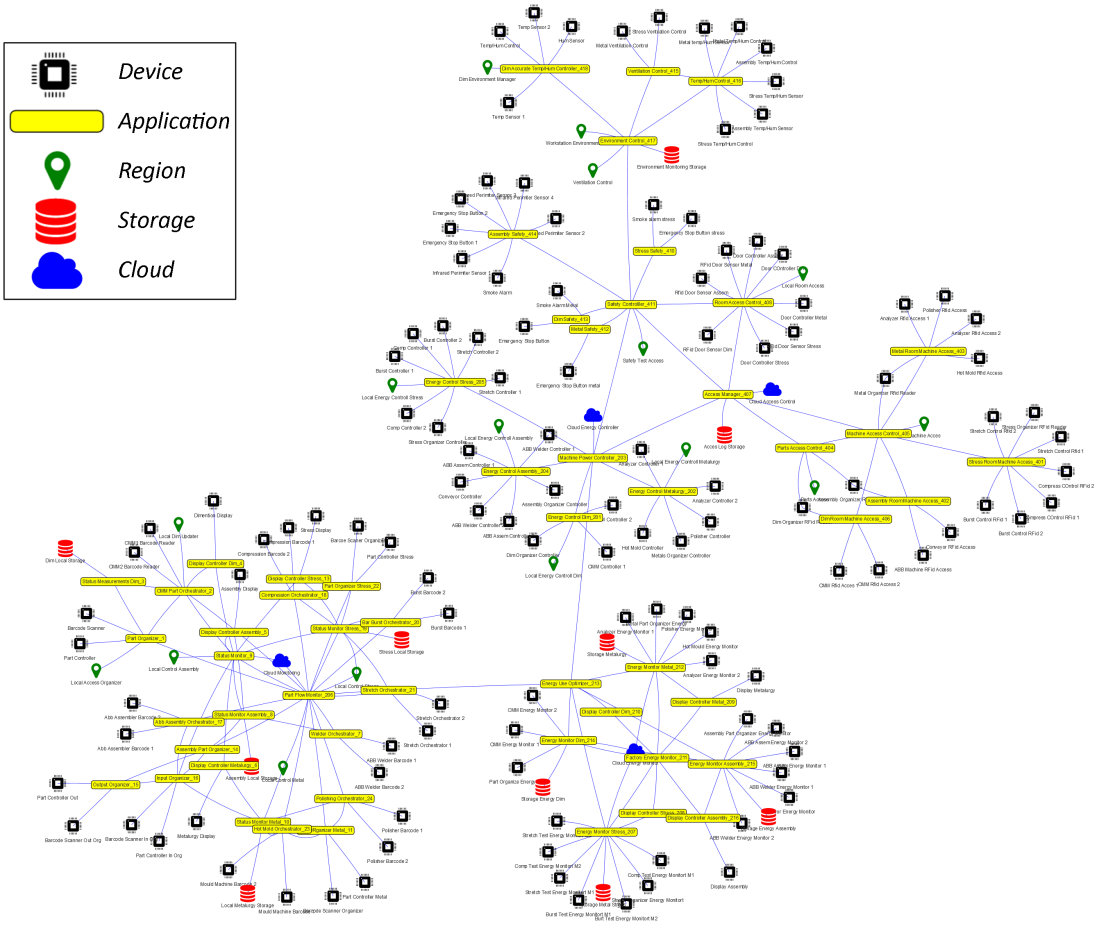

Fig. 6.4 Combined System

sets. The second reason is to identify the characteristics of these systems that can be used to better select and create new optimisation approaches. Finally, as proposed in (Voutyras et al. 2015) these parameters can be used to calculate or estimate latencies, reliability and redundancies of entities and the system. For the analysis, the system is considered a graph $G = (V, E)$ where V denoted the vertexes, nodes denote the applications, storages, Cloud entities, and regional access points while E denotes the Edges or connections between these.

Considering $V_i \in V_k \in V$ where k denotes the type of Node and i denotes the number or id of the node and V_k denotes the set of all nodes of the same type. The edges are denoted by $E_{i,j} \in E$ where i and j are the id of the connected Nodes and $E_{i,j}$ denotes the edge itself where $E_{i,j} = E_{j,i}$ due to the undirected and unweighted nature of the graph.

6.1.2.1 Replication Parameters

The replication parameters are simple properties of the graphs that look at key parameters that are used to replicate the structure of the graph to allow for the scaling of certain use-cases.

Application Resource Use looks at what is the average number and distribution of device, region, storage and Cloud connections from applications. The resource use of an Application is denoted by R_{App}^{Type} where Type is the resource type and App is the application id. Eq. 6.1 defines this resource use as a sum of all connections of an application to a type of device.

$$R_{App}^{Type} = \sum_{i}^{E_{App,i}} V_i \in V_{Type} \tag{6.1}$$

Clustering of Applications looks at how certain applications group together into clusters and what is the average size and number of these clusters and how interconnected they are. Cluster $Clust_i$ is defined, where Application $V_i \in Clust_i$ as defined by a clustering algorithm like K-Means or DBSCAN (R. Xu and Wunsch 2005).

Connection Locality looks at what are the chances of one application connecting to resources and devices from the same gateway and how many external resources and applications it uses. The distribution of these types of connections is crucial to replication. Three types of locality $\{Local, Cluster, External\}$ are defined, where $V_i \in Loc_k^{Local}$ so that all elements V_i are on the same gateway, $V_j \in Loc_k^{Cluster}$ so that all elements V_j are part of the same Cluster and $V_l \in Loc_k^{External}$ as in Eq. 6.2.

$$Loc_k^{External} = V - (Loc_k^{Local} \cup Loc_k^{Cluster}) \tag{6.2}$$

Inter-App communication looks at the average number and distribution of connections between applications deployed on the system. Together with clustering and locality, this component helps create a more realistic environment. The connections of an Application are denoted by C_{App}^{Area} where $Area$ denotes the region to which the application connects to which can be local or cluster level. Eq. 6.3 defines these connections as a sum of all connections from each region coming or going to the application.

$$\begin{aligned} C_{App}^{Area} &= \sum_{i}^{E_{iApp}} V_i \in Loc_k^{Area}, \; where \; V_{App} \in Loc_k^{Area} \\ C_{App}^{Ext} &= \sum_{i}^{E_{iApp}} V_i \in Loc_k^{Ext}, \; where \; V_{App} \notin Loc_k^{Ext} \end{aligned} \tag{6.3}$$

6.1.2.2 Graph Parameters

The graph parameters are designed to show certain characteristics of these systems that can be translated to parameters of interest, such as reliability, latencies, clustering and interconnectivity. These characteristics are used in (Newman 2003) to analyse a varying range of systems such as the World-wide-web, social networks, citation interconnectivity and others.

Connectivity checks if there is a route $route(i, j)$ from any node V_i in the graph to any other node V_j in the system. After verifying connectivity, the distinct connected graphs are analysed. This parameter aids in clustering of these connected graphs as well as shows separate subsystems. The use-cases are all connected graphs so this parameter, while important in the analysis, in the case is overlooked when discussing results.

Average path lengths look at what is the average distance between two nodes while the graph diameter looks at the maximum distance. These parameters can be used to determine simple average and maximum latencies and hops within a network while comparing them to node and vertex counts can help determine QoS parameters. The minimum distance from node V_i in the graph to any node V_j can be computed through the Dijkstra's algorithms and is denoted with $route_{Min}(i, j)$ while the average path in a system is defined as in Eq. 6.4 and the diameter or maximum shortest route is defined in Eq. 6.5.

$$AVG_{Route} = \frac{\sum_i^V \sum_j^V route_{Min}(i, j),\ where\ i \neq j}{size(V)(size(V) - 1)} \tag{6.4}$$

$$Diameter = \max_{V_i \in V} (\max_{V_j \in V} route_{Min}(i, j)),\ where\ i \neq j \tag{6.5}$$

The clustering coefficient looks at the average number of triangles $Tri(V_i)$, or three node pairs with each node being a member of a system. This number is divided by the total number of possible triangles, adjusting for the size of the graph. The sum of these values is the Clustering Coefficient (CCF) of the graph as can be seen in Eq. 6.6. This information can be used to determine how tightly coupled a cluster is. This parameter could be useful in determining the optimisation of subsystems using divide and conquer techniques in optimisation, especially latency optimisation.

$$CCF = \sum_i^V \frac{Tri(V_i)}{size(V)(size(V) - 1)} \tag{6.6}$$

The graph degree distribution (GDD) looks at how many nodes have a certain number of connections in a system compared to the maximum possible number of connections. The

number of nodes that have a certain degree can be calculated based on Eq. 6.7 where k is the edge count.

$$GDD[k] = \sum_{i}^{V} \sum_{j}^{E} E_{i,j} = k \tag{6.7}$$

This gives a view of how the connections differ between systems and can provide the main comparison factor when categorising the system as well as verifying generated systems.

The Graph Betweenness Centrality (GBC) of a node is calculated by counting the number of shortest paths $route_{Min}(i, j)$ that contain a node and compare it to the maximum and minimum values present in the system. This implementation looks at all the paths whose length is equal to the shortest one. The Eq. 6.8 shows how the centrality of one node is calculated.

$$GBC(V_i) = \sum_{j}^{V} \sum_{l}^{V} V_i \in route_{Min}(j,l), \; where \; i \neq j,l \tag{6.8}$$

The distribution looks at how many nodes have these values between a certain range. This parameter is key in determining high importance nodes in the system as well as critical single points of failure. This characteristic is also important when comparing systems and verifying the generated graphs.

6.1.2.3 Network based Categorisation

There are several network types based on their connection typology as suggested in (Newman 2003), each having their real-world equivalent and their set of attributes. The use-cases are then compared to the behaviour of known models such as random-graphs, Markow graphs, non-scalable networks, small-world models, Barabas-Albert and other growth models.

With each network having its own characteristics, they require different approaches when certain optimisation or analysis attempts are made such as clustering and single point of failure rerouting.

The analysis and categorisation approximation of the system will allow for model specific method to be applied which may reduce run-times and reduce the diminishing returns seen in similar systems, such as in (Heller, Sherwood, and McKeown 2012).

6.1.3 Replication Data Analysis

The data analysis for the 4 virtual scenarios from the application resource use and locality point of view can be seen in Table 6.1. broken down to device, storage, Cloud and local

interfaces and computed through the equation in Eq. 6.2. Each component has a local and external factor which looks at the locality of these connections with the local being the gateway hosting most resources while the external represents other gateways.

The connections between application are described in Table 6.2. Where they are broken down to local connections, cluster connections and external connections based on Eq. 6.3 and Eq.6.4. These are important when designing systems when considering approaches that focus on connections remapping SDN based router rewiring and other similar methods.

The clustered connections refer to the clusters in Fig. 6.5 and looks at all the connections that are not to the same Gateway but are in the same cluster, while the external ones look at all connections to external gateways not on the cluster while the total shows all the connections.

Table 6.1 Resource Use Parameters

Prop	Scenario											
	Energy			Parts and Flow			Access and Security			Combined Systems		
	Loc	Ext	Tot	Loc	Ext	Tot	Loc	Ext	Tot	Loc	Ext	Tot
Device												
Min	0	0	0	0	0	0	0	0	0	0	0	0
Max	7	0	7	2	0	2	8	1	9	8	1	9
Avg	2.87	0.0	2.87	1.25	0.0	1.25	2.94	0.05	3.0	2.3	0.01	2.32
Cloud												
Min	0	0	0	0	0	0	0	0	0	0	0	0
Max	1	1	2	1	1	2	1	0	1	1	1	2
Avg	0.12	0.18	0.31	0.04	0.04	0.08	0.05	0.0	0.05	0.07	0.07	0.14
Storage												
Min	0	0	0	0	0	0	0	0	0	0	0	0
Max	1	0	1	1	0	1	1	0	1	1	0	1
Avg	0.25	0.0	0.25	0.16	0.0	0.16	0.11	0.0	0.11	0.17	0.0	0.17
Local Access												
Min	0	0	0	0	0	0	0	0	0	0	0	0
Max	1	0	1	1	1	2	2	0	2	2	2	4
Avg	0.25	0.0	0.25	0.2	0.08	0.29	0.38	0.0	0.38	0.26	0.05	0.32

Determining the number and size of the clusters for the analysis that was used for the app data in Table 6.2. was done using a Density-Based Clustering Scan (DBSCAN) on the graphs.

The configuration of the scan requires a minimum number of points for a cluster which for this case is 8 and an epsilon which is a maximum distance between two peers which in the graph is 1. The minimum points value is determined by the structure of the graph. A

Table 6.2 Application Parameters

Property	Parameters			
	Local	Cluster	External	Total
Energy Monitoring and Controll				
Min	0	0	0	0
Max	4	6	1	8
Average	1.375	1.25	0.125	2.75
Parts and Flow Monitoring				
Min	0	0	0	2
Max	5	11	2	15
Average	2.0	1.5	0.33	3.83
Access and Security Control				
Min	0	0	0	1
Max	6	4	1	8
Average	1.66	0.55	0.11	2.33
Combined System				
Min	0	0	0	1
Max	6	14	3	15
Average	1.64	1.21	0.32	3.17

more highly connected graph would require higher values to return distinct clusters rather than one big cluster.

The resulting clusters can be seen in Fig. 6.5 where (a) is the Parts and Flow Monitoring system, (b) is the Access Safety and Environmental Control and monitoring subsystem, (c) is the Energy Monitoring optimisation and Control subsystem and (d) is the Combined System. Individual application clusters are coloured the same and applications that are not part of any cluster are coloured white.

This clustering method made for an average cluster size of 7.42, a maximum of 22 and minimum of 1. The method resulted in an average of 2.25 applications not being assigned a cluster. This method works well in (d) and (b) where the density of nodes is more uniform and the results are weaker in (a) where the tightly coupled nature of applications results in one big cluster. In (c) due to the varying density, the top part of the graph is well clustered while on the bottom it identifies two small clusters and two unassigned nodes.

6.1.4 Network Analysis

The subsystems are analysed based on the parameters in the previous section where the connectivity path length and diameter are the more basic properties of the system. For these tests, all the systems are made up of connected graphs, but this test would allow a fast

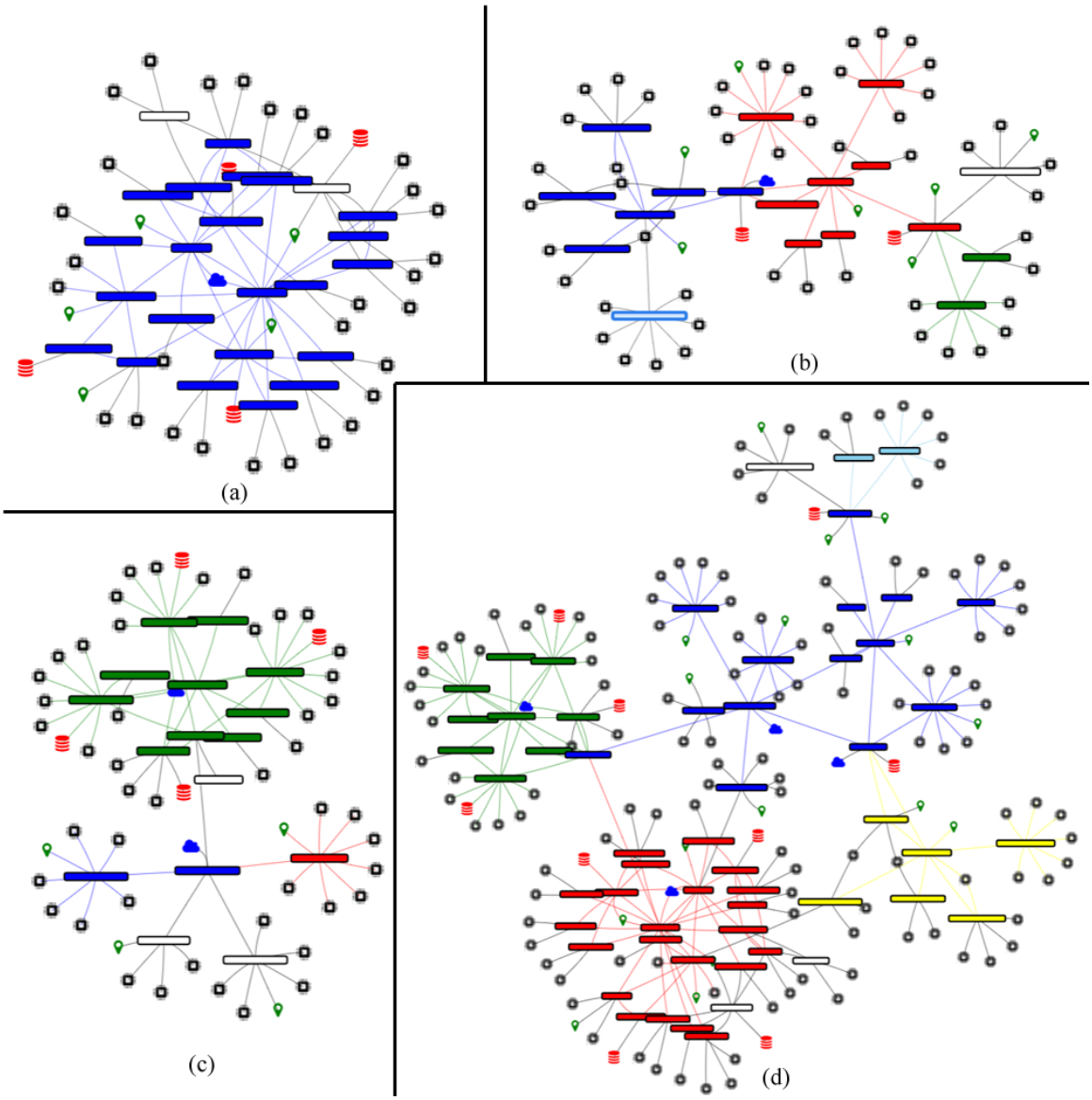

Fig. 6.5 DBSCAN Clustering Results

clustering and easier group based optimisation in cases such as the combined system if there were no connections between subsystems. The average diameter is 7 hops, while the average path length is 4.15. The maximum diameter is in the combined system with 9 as well as the highest average path length of 5.23. The diameter and average path length (APL) increases with the size of the cluster and are reduced with the increase of clustering as in (c) with a Clustering Coefficient (CCF) of 0.01 having an APL of 3.84 and the more tightly clustered (a) with a CCF of 0.09 has an APL of 3.29.

Looking at the CCF of the applications on not just the systems but also on the subgraphs. The average CCF of the systems is 0.0425 varying between 0.016 and 0.09. If the clusters

are taken by themselves the average CCF of clusters that have a size larger than 2 is 0.208 with values between 0.09 and 0.46.

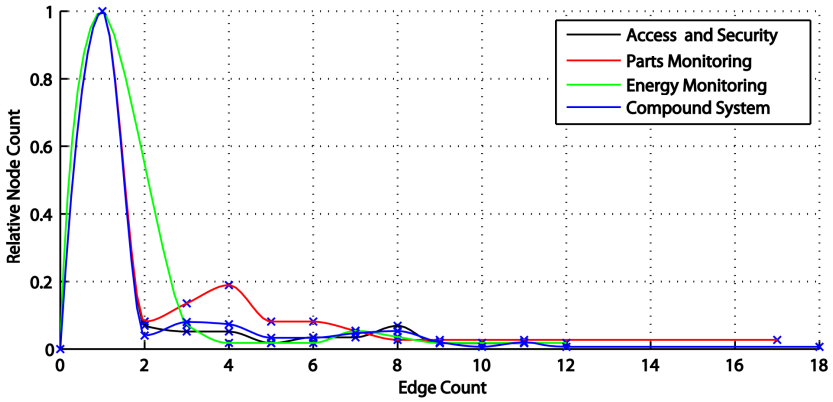

Fig. 6.6 Graph Degree Distribution of Systems

The Graph Degree Distribution of the systems can be seen in Fig. 6.6. The number of nodes displayed is relative to the maximum number of nodes to allow a comparison between the graphs. For the systems, the highest node count values were at 1 connections, which is due to the device and resource links which are usually used by one application. The maximum values for these are 58 for Access in Fig. (6.5.b), 37 for parts monitoring in Fig. (6.5.a), 56 for Energy in Fig. (6.5.c) and 150 for the combined system in Fig. (6.5.d). The highest number of edges are on the combined system with 18 and the second is on the Parts monitoring with 17. Every Node has at least one connection as the connectivity of the graphs show as well.

The Graph Betweenness of the systems is shown in Fig. 6.7. The centrality value is a relative value to the maximum available on the system which is scaled to account for network size differences. The relative node count is scaled to the max values as well.

The node in (d) with the highest absolute centrality has a value of 40745 possible shortest paths crossing this node. This high number is also due to the implementation of the algorithm where the minimum distance between two nodes is calculated and all paths of the same lengths are considered. These values are 3763 for the Energy Monitoring, 4012 for Access Control and 3886 for Parts Monitoring. The devices and resources often have a value of 0 residing at the edge of the network, not providing a connection between any two components.

Based on the betweenness data as well as the graph degree distribution and structure of the system it can be shown that there are some similarities with existing models. The Access Control and Energy Monitoring Systems have similar structures and the data in Fig. 6.6 and Fig. 6.7 show that they have similar properties in structure to hierarchical and

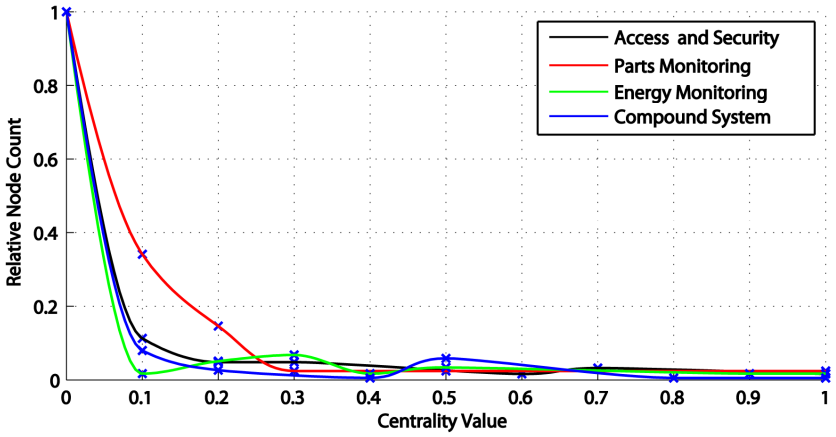

Fig. 6.7 Graph Betweenness Distribution of Systems

fractal networks with certain outliers and density variations. A closer look at these systems shows that their distribution and betweenness, especially that of the Access Control are like a Barabási-Albert model with an initial degree, $m_0 = 1$. The Parts Monitoring system has a different architecture with similar properties to a Random Graph when looking at the application's connections and the lack of clustering, as well as the outliers in Fig. 6.5. and Fig. 6.6. For the Combined system, the plotted data, as well as its structure, suggest that it has similar attributes to the Random Network that models the World Wide Web (WWW), having clusters form and a varied type of connections.

6.1.5 Replication Analysis

When looking at the parameters used to generate use cases, certain properties of interest are considered. The increased adoption of connection locality and clustering can be seen in Fig. 6.8. Part (a) shows a completely random system with just the node numbers and average connection data being used. Part (b) adds connections types, distribution and locality, while part (c) adds the remaining factor of clustering.

The data in Fig. 6.8 shows that as more parameters are adopted the systems resembles those presented in Fig. 6.5. The system in (b) is similar to the Parts Monitoring use-case with the exception that devices are more interconnected due to the lack of locality data. System (c) contain all considered parameters and is like the Combined use-case and the Energy Monitoring one. When considering even more realistic systems, the scaled generations based on Random Networks or Barabási-Albert models should be considered. These can form the basis of how these will scale up.

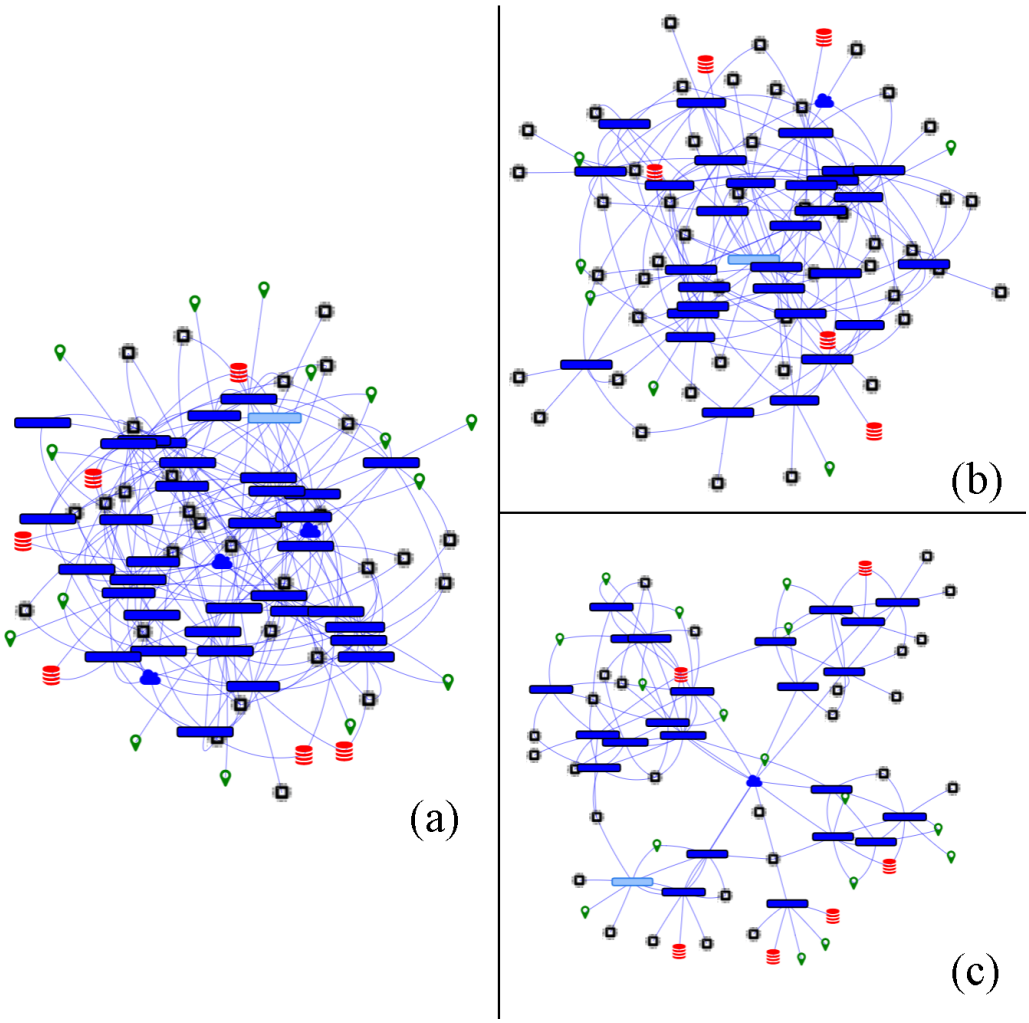

Fig. 6.8 Replicated Systems

6.2 Model Validation

For these tests, a set of applications are proposed that have different loads and message rates. These are tested individually on the gateways. They are deployed in groups and migrated to verify that the presented application model stands and that the estimations for Gateway Load and Application Delay are correct. Based on these the accuracy of the model is measured. After the model is verified and the resulting parameters tested, two sets of optimisation tests are performed.

The first set consists of testing the optimisation of a system of four gateways having a varying set of applications deployed on them. The run-time parameters are measured, validating them to the model, then the optimisation algorithms are run, deploying the results

and validating them on the physical system. Based on these tests two approaches are evaluated on a small-scale deployment.

In the second set, the scalability of the methods and optimisation algorithms are tested by generating a random set of gateways with a set of available Cloud VMs with random latencies between them and a random set of deployed applications and look at how well each method performs.

The main assessment criteria for the model are the precision of the Load and Delay estimation on the system. The optimisation methods are assessed by looking at the decrease in application delays and the reduction of constraint violations that are adjusted to the size of the cluster by looking at how much the total delay drops on average for each gateway as well as how many applications can fulfil their requirements on average per gateway.

6.2.1 Single Deployment Validation

The initial model's error is known from the fitting tests which resulted in 7.34%. A more extensive test is performed and the results evaluated when deploying to four RPi1 type gateway and an equivalent Cloud GW. There are 8 applications with varying loads of 0.465, 1.45, 4.265 and 9.915 and varying message rates of 1, 4 and 20.

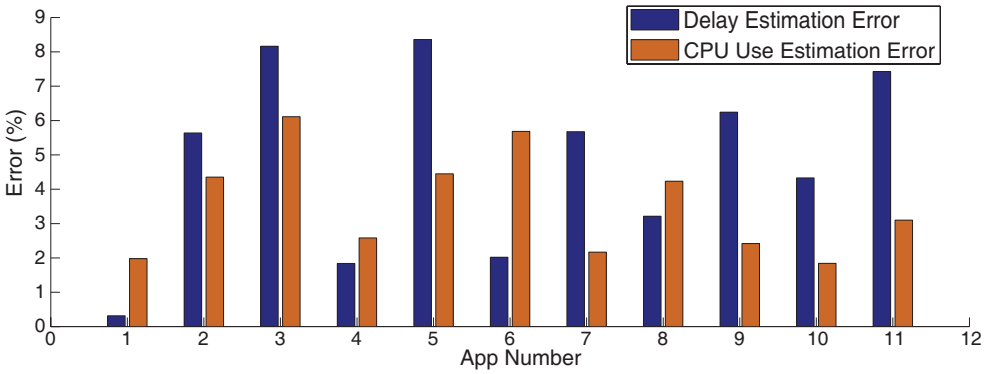

Fig. 6.9 Single Deployments Results

This test consists of deploying applications on a free gateway, measuring the app and the gateways run-time parameters and comparing them to the estimations that the application model made. The results of these tests can be seen in Fig. 6.9.The Idle Load L_j^{Idle} in Eq. (4.4) for these tests is 5.64. The estimations of a single app deployment had an average error for the Total Load of 3.30% and 4.99% for the Estimated Delay, with a maximum error for the load of 5.68% and 8.35% for the delay. These maximum values were achieved at the edges of the linearization by two apps. For further testing, the remainder of 9 apps are considered that lacked the maximums.

6.2.2 Bundled Deployment Validation

Bundled deployment testing considers the applications from the first test and deploys 10 sets of them on the gateway and verifies how accurate the model is in determining the application delay and the total processor use. For these tests, applications have the same Unit Load as in Eq. 4.5. The accuracy of these deployments can be seen in Fig. 6.10.

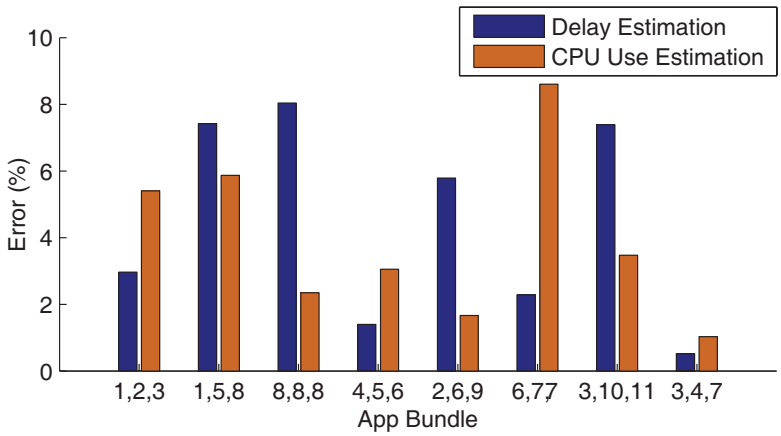

Fig. 6.10 Bundled Deployments Results

From these tests, the Gateway Load estimation error when deploying a set of applications is 3.92% while the maximum value is 8.6% which was found for the deployments with applications of low message rates. The Delay estimation error was found to have a mean of 5.47% and the worst result of 9.04% from the same set of applications.

6.2.3 Migration Deployment Validation

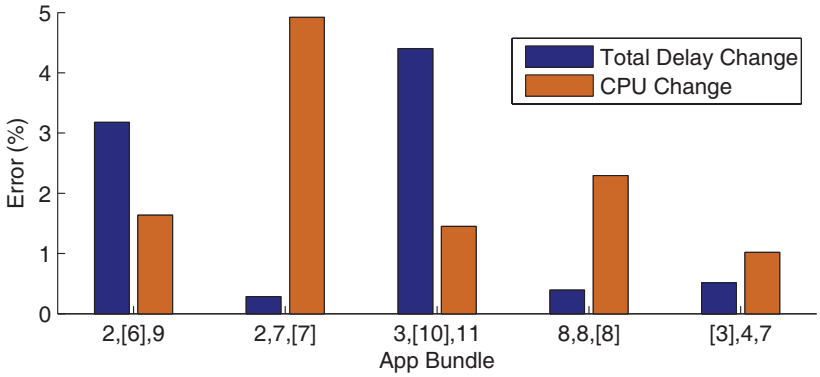

Fig. 6.11 Migration Deployments Results

For the final tests, an application is migrated while measuring the changes in the original host of the application and considering the delay time of the application and how accurate this estimation was. The results can be seen in Fig. 6.11.

The resulting errors measure the estimation error increased with the total value of the delay. In these cases, the average estimation error of the CPU of 2.26% with a maximum of 4.91 was found. The delay estimations had an average error in accuracy of 1.75% with a peak of 4.4%. This is partially due to the small size of the errors compared to the total delay as well as using monitored parameters to estimate the results of the migration.

6.3 Physical System Deployment Optimisation

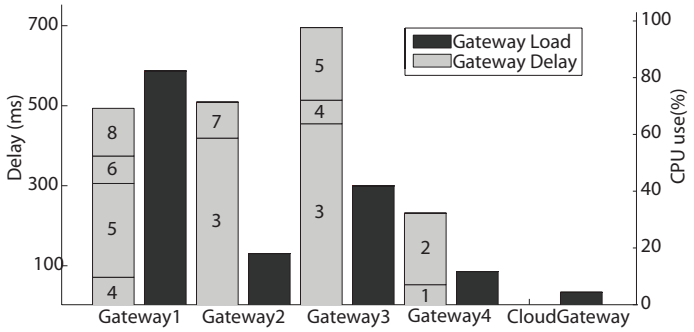

Fig. 6.12 Initial Deployment

For these tests the Fog of Things Platform has 4 gateways and a Virtual Cloud Gateway. The gateways have a delay between them of an average of 19.53ms while the delay between the Cloud gateway is set to an average of 42ms. The initial state of the Cloud is empty only having application migrated to it after optimisation if needed. A set of initial configurations are generated, deployed and monitored on the gateway. After this, the information is fed into the optimisation algorithms that come up with the best solution within the parameters and proposed functions.

The results are then deployed on the physical cluster and the actual values are examined. The initial Delays and CPU usage of the gateway can be seen in Fig. 6.12. The total delay of the system is 1881.80ms while the average Load variation is 21.67% with a System Reliability of 66.53% and there are 2 applications that do not meet their constraints.

For this initial phase, the differences in results from the given fitness functions are of the highest interest. For this set, the best results from the Hungarian, GA and Random methods are used. The results can be seen in Fig. 6.15 for Reliability Optimisation and Fig. 6.13

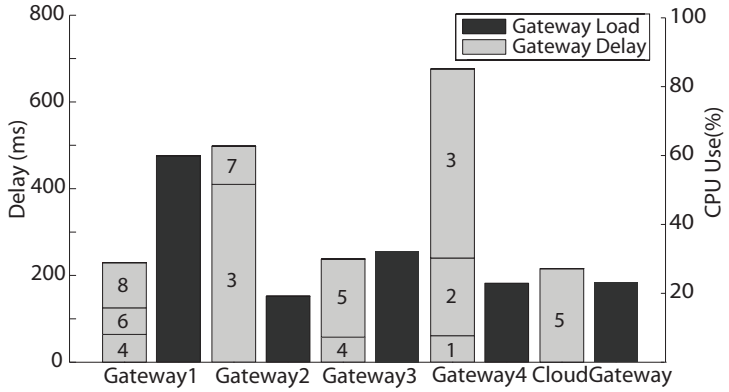

Fig. 6.13 Delay Optimisation

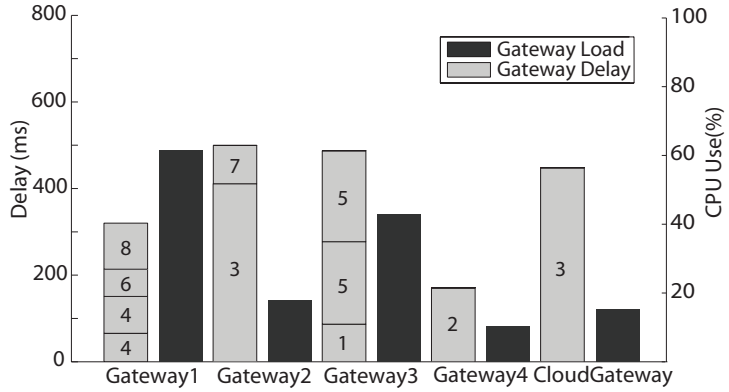

Fig. 6.14 Constraint Delay Optimisation

for Delay Optimisation and Fig. 6.14 Constraint and Delay Optimisation. Here the number represents the application ID.

From these tests, the Reliability Optimisation method managed to reduce the load variation to 2.07% and had the maximum reliability of 73.56%. This did not improve constraint violations and it actually increased the delays to 2038ms due to unnecessary migration. The Delay Optimisation achieved a minimum total delay of 1854ms with 1 constraint violation while the Constraint and Delay Optimisation had a higher total delay of 1974.1 ms but managed to have 0 constraint violations. Both the Delay Optimisation and Constraint and Delay Optimisation methods improved the Reliability to 72.35% and 71.43% which is an improvement to the initial deployment but falls short of the Reliability Optimisation results.

These tests show that some methods that are tailored to solve single parameter problems affect other parameters as well, in some cases improving them. Due to the exponential nature of the reliability function, the best overall reliability was achieved by balancing out the applications on the system. This solution had the consequence of creating the worst

overall delay on the system. These physical deployments tests serve as proof of the need for multi-parameter methods but also show the interconnected nature of these parameters.

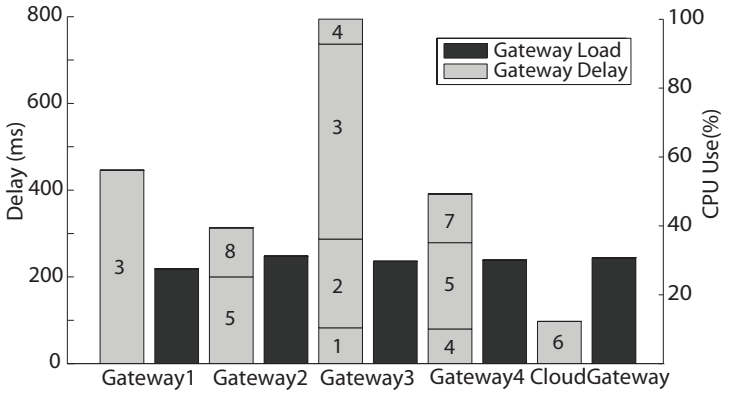

Fig. 6.15 Reliability Optimisation Results

6.4 Evaluation Use Cases

The evaluation use cases are deployment scenarios that are designed to evaluate the performance of the algorithms in increasingly difficult conditions. The difficulty is increased in two directions, the first direction is the validation, where the deployed applications require more parameters to in line so their deployment is acceptable. The second direction is the complexity of the utility function. In the first case, a single component utility function is examined while later on varying weights and constraints based utility is considered as well.

Three scenarios are considered. The simplest one, with the most basic requirements, is the delay optimisation which only looks at reducing the delays on the system. The Weighted Multi-component utility scenario increases the complexity of the optimisation tasks by adding multiple parameters that need to be improved as well as adding soft constraints to the system. The final scenario adds capabilities and requirements matching to the equation which makes validation more difficult.

The scaling and expansion functions of these scenarios, when generating test cases are based on the parameters found in Table 6.1 and Table 6.2 that generates Fog systems that resemble the one presented in Fig. 6.5 (d).

6.4.1 Delay Optimisation Scenario

The delay optimisation scenario is the most basic, as it considers no soft or hard constraints and looks at improving only a single parameter, the system delay. The system delay is defined

as in Eq. 4.7 and the utility function can be seen in Eq. 6.9 as the sum of all the application delays on Fog F.

$$Util^F_{Delay} = D^F = \sum_{i=0}^{n} D^F_i \tag{6.9}$$

The weights for the delay W_i^{Delay} is 1.0 for all the applications. Validating a deployment in this scenario is simply the case of going through each gateway and checking if their maximum load value has exceeded the maximum allowed, which is 99% of its load capability.

6.4.2 Weighted Multi-Component Utility Scenario

The weighted multi-component Utility scenario proposes a multi-property utility that is designed to create a more difficult optimisation problem. In this scenario, the total utility of the system and an individual can be seen in Eq. 6.10 .

$$Util^F_{Multi-Comp} = D^F + R^F + Ct^F = \sum_{i=0}^{n} D^F_i + R^F_i + Ct^F_i \tag{6.10}$$

Here, D^F_i is calculated as in Eq. 4.17, R^F_i is done as in Eq. 4.19 and Ct^F_i is computed as presented in Eq. 4.21. When considering these scenarios, the weightings of the applications W^X_i are modified as well so certain applications are more interested in one parameter over another having $W_i^{Constraint\_Violations}$ at 1.0 when they are present and at 0.0 when not and having W_i^{Delay} and $W_i^{Reliability}$ in the range $\{0.0, 0.33, 0.66, 1.0\}$. In these tests, there is a 0.2 chance that an application has soft constraints.

The soft constraints for the application reliability and delay are generated by deploying the application to a theoretical gateway that contains the average system load, adding 10% and considering its delay and reliability in that case as the reference. The soft constraints don't affect the validity of a deployment as their impact is solely on the total utility. This allows the use of the same method for validation as for the Delay optimisation scenario above.

6.4.3 Capability Constraint and Utility Scenario

The Capability constraint and Utility scenario builds on the previous case where the same utility from Eq. 6.10 is used and the same method for generating new test cases. The only difference between this and the previous scenario is the addition of gateway capabilities Cap^{Gw}_j and application requirements Cap^A_i. These are considered hard constraints, which

means if they are not satisfied, the resulting deployment solution is not valid. This further increases the difficulty of the optimisation problem that is being tested.

In this setup, 7 possible Capabilities are considered, out of which each gateway contains at least 4 different ones and each application requires only one Capability from the gateway. This creates a more difficult validation scenario than for the previous scenarios where the gateway load limit was the only restrictive factor.

6.5 Testing Parameter Selection

When considering the testing parameters for the validation section, these parameters can be put into two categories. The first category of parameters influences the results of the optimisation methods in a limited way but doesn't determine whether these will terminate. The values for these can be seen in Table 6.3. and are considered constant during the testing and are tuned to the existing parameters based on the literature available for the scenario generations and based on the breadth of tests that were performed during the development for the method parameters.

The test was deployed and run on a Spark Cluster deployed on five workers that have the configuration and processing capability of VM1 from Table 4.1. Each test is allocated 2GB of Ram for the worker and 1GB of ram for the Driver and 1 CPU each.

Besides these fixed parameters there are those that can greatly influence the results of the optimisation methods. The tuning of these values changes with the scenario selection as well as with deployment size. The tuning of the eps parameter is automatically done by one of the methods as it's tuning values differ too much between scenarios and chosen weights and would not be practical.

6.5.1 GA Parameter Selection

When considering the dominant parameter selection for the GA, other than the parameters that are fixed from the previous section, the generation size and the stopping condition need to be set which are considered by (Boyabatli and Sabuncuoglu 2004) to require specialist knowledge and there is no clear way exists of finding these.

For the generation sizes, a number of tests are run in Fig. 6.16. for the Delay Scenario, in Fig. 6.17. for the Multi-Parameter scenario and in Fig. 6.18. for the Capability Scenario. The resulting scaling formula for these methods can be seen if Eq. 6.11.

The graphs were generated for each scenario in the same way with the Fog Size being the only varying parameter as the Capability Scenario is too difficult for the GA to solve at

Table 6.3 Fixed Method Parameters

Parameter	Used In	Value/ Range
Inside Cluster Latency	Scenario Generation	8.97-30.89
Inside Cloud latency		2.37-6.89
Edge Cloud Latency		37.37-87.89
Cloud Edge Gw Ratio		0.1
Max App Load		30%
Max Fog Load		60%
Constraint Allocation Rate		0.1
Constraint Improvement		8%
Edge Gateway Capacity		1.0-1.4
Edge Gateway Speedup		1.0-1.4
Cloud Gateway Capacity		1.8-2.5
Cloud Gateway Speedup		2.8-4.3
Random Population	Genetic Algorithms	0.4
Elitism Population		0.2
Crossover Population		0.2
Mutation Population		0.2
Mutation Chance		0.1
Eps	Distance Clustering	1
minPts		7
Max Gw Division	Resource Allocation	2
Gw Share Threshold		0.3
Cluster Allocation Cutoff		0.8
Min Allocation Req		1.2
Max Fail Count	Iterative Weights Calculation	3
Max Iteration Count		10
App Consideration Limit		0.2
Gw Consideration Limit		0.05
Better Solution Limit		4%
Penalty Value		0.2
Consideration Change Rate		10%
Sample Relative Size	Sampling Weights Initialisation	0.1
Min Sample Size		10
Size Increase Multiplier		2
Size Decrease Multiplier		0.9
Max Fail Count		10

the scale the other scenarios were tested. The legends of the figures show the Fog Sizes that were used, while the Polyfit represents the dotted line that shows how the size needs to be adjusted as the system is scaled.

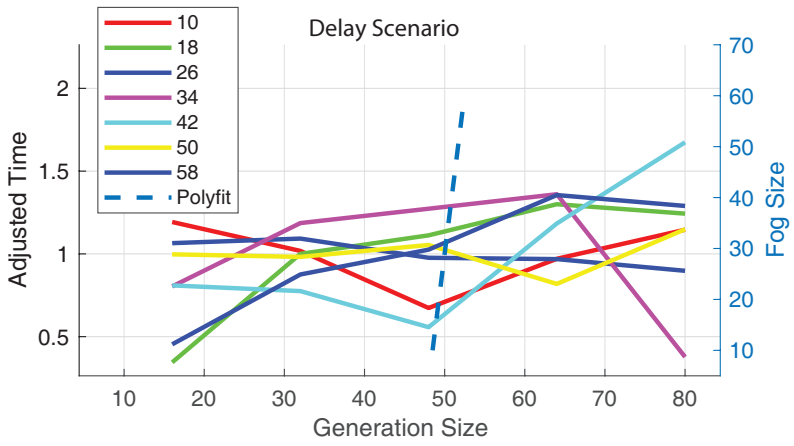

Fig. 6.16 Generation Size Variation Impact - Delay

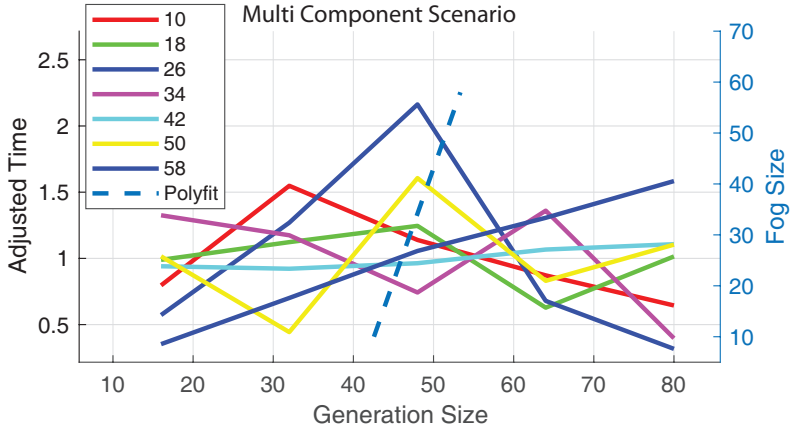

Fig. 6.17 Generation Size Variation Impact - Multi-Parameter

The values for the Adjusted time are calculated by first generating a Fog Deployment and using a GA with a generation size of 200 and an iteration size of 5000 to find a reference utility function. Then the generation size is then varied within the shown parameters and repeated 5 times while the average time to reach the reference utility is recorded. When plotting these values are adjusted so they don't represent time but rather their deviation from the mean time for that instance.

The *PolyFit* values and equations are derived by multiplying the adjusted time with the respective generation size and dividing it by the sampling rate. If these values are considered for each Fog Size and then a linearization method is used, they result in Eq. 6.11 where the F_{Size} represents the Fog Size.

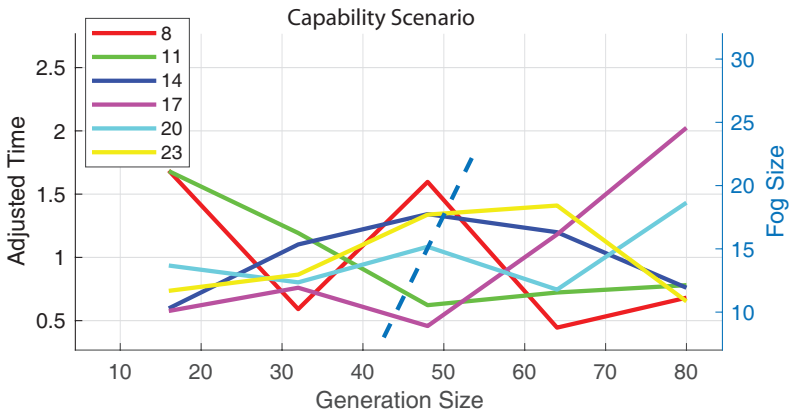

Fig. 6.18 Generation Size Variation Impact - Capability

$$\begin{aligned} Delay\ Scenario &: 47.69 + 0.079 F_{Size} \\ Multi-Parameter\ Scenario &: 40.31 + 0.22 F_{Size} \\ Capability\ Scenario &: 36.43 + 0.76 * F_{Size} \end{aligned} \tag{6.11}$$

When considering the stopping condition for the GA algorithms used in the validation a simple *maxIteration* count would not allow the system to find the best solutions, so a different method is proposed where the GA terminates if it has been stagnant for a number of iterations with no improved utility found. For this stagnation value, the double of each scenario's largest stagnation point is considered.

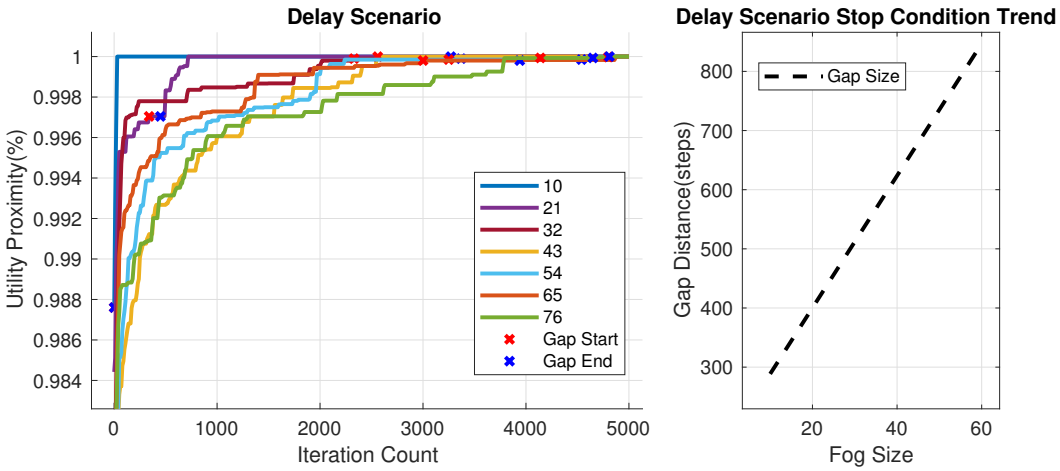

Fig. 6.19 Stop Condition - Delay

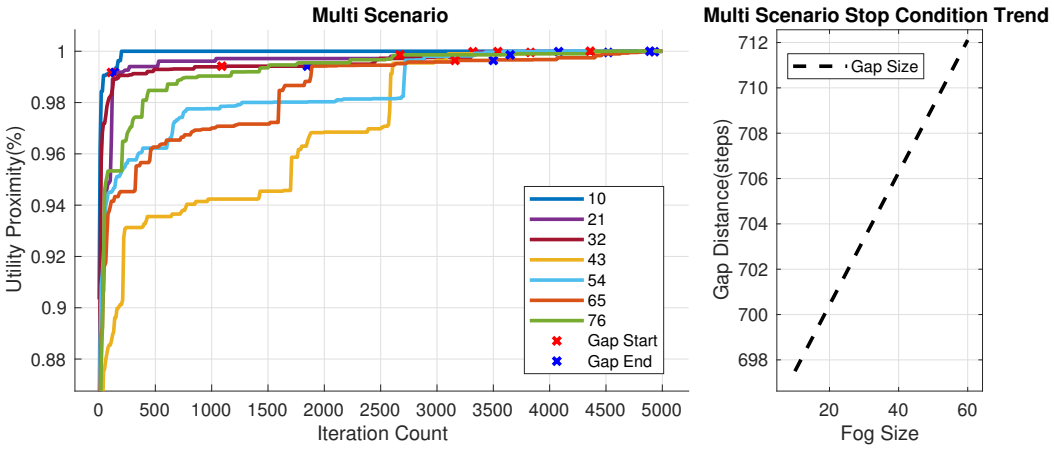

Fig. 6.20 Stop Condition - Multi-Parameter

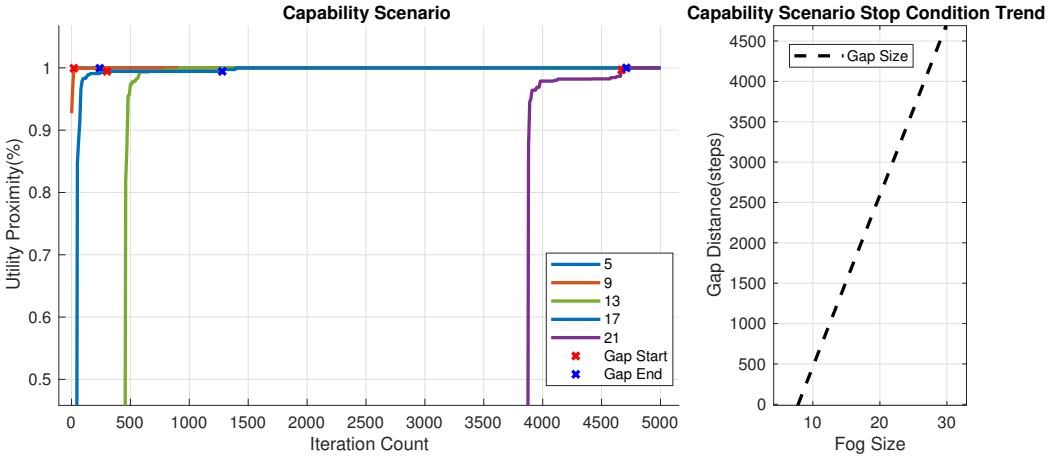

Fig. 6.21 Stop Condition - Capability

For these stopping conditions, a number of tests are run in Fig. 6.19. for the Delay Scenario, in Fig. 6.20. for the Multi-Parameter scenario and in Fig. 6.21. for the Capability Scenario. The resulting scaling formula for these methods can be seen if Eq. 6.12.

$$\begin{aligned} \textit{Delay Scenario} &: 176.88 + 11.168 F_{Size} \\ \textit{Multi} - \textit{Parameter Scenario} &: 694.57 + 0.2922 F_{Size} \\ \textit{Capability Scenario} &: 178.45 * F_{Size} \end{aligned} \tag{6.12}$$

The figures were generated in the same way, with only the Capability Scenario receiving lower thresholds for the Fog Size as in the previous tests. The legends show the size of the tested Fog environments while the figures show how they reached the best results and what

the utility values were at each iteration. The *GapStart* and *GapEnd* points show where the major gaps were in the testing. The adjacent figure shows the trend in the gap sizes with the increase in scale. The iteration counts were fixed to 5000 for these scenarios.

The Utility proximity values are calculated by comparing each utility to the best one the method found. The Gap Sizes were calculated by finding the largest stagnation periods for each iteration excluding the one leading to the 5000th iteration and linearizing these for each scenario resulting in the equations in Eq. 6.12. The equation for the Capability Scenario was adjusted as to start from 0 and have a slope a of 178.45 as from the linearization, the starting point or value of b was found to be -1658.3 which would result in small scales having a stopping condition that is negative.

6.5.2 Clustering Parameter Selection

Selecting the right clustering parameters can determine the quality of a test and also its run-time. Higher Cluster sizes mean that the search-space is larger which should allow for better minimum points to be found but can also lead to the methods getting stuck or not being able to meaningfully cover the space. A lower cluster size while improving the time-efficiency of the algorithm might result in situations where the search-space is so small there is no valid solution or there is little to no room for improvement.

The attempt to get a reference guide for these sizes to be able to rule these parameters out when evaluating the methods can be seen in Fig. 6.22. where the three scenarios can be seen in one graph with they approximated quadratic functions.

The test was conducted by generating a Fog environment of a certain size and scenario and running a GA Optimisation on this using the stopping conditions specified in Eq. 6.12. If there was no valid deployment found a new Fog was generated and the test was re-run a maximum of 5 times after which a value of 0.0 improvement for the test is noted. The utility improvement is calculated by getting the best utility of the first randomly generated population and comparing that to the best of the last generation. If there is no improvement, a value of 0.0 is noted as in the previous case.

Based on these tests it can be shown that increasing the difficulty of the problem reduces the range of Fog Sizes where the GA works well. The Multi-Component scenario has a steeper slope than the Delay scenario and the Capability Scenario has the steepest slope and smallest Fog Size Range.

Setting the *minPts* value for the Clustering doesn't directly results in cluster sizes but from the conducted tests, the size of the resulting clusters is also determined by the confidence factor of the parameters but a reference value of 0.25 of the desired size can be considered as this allows the initial clusters which are based on less reliable weights to be found and

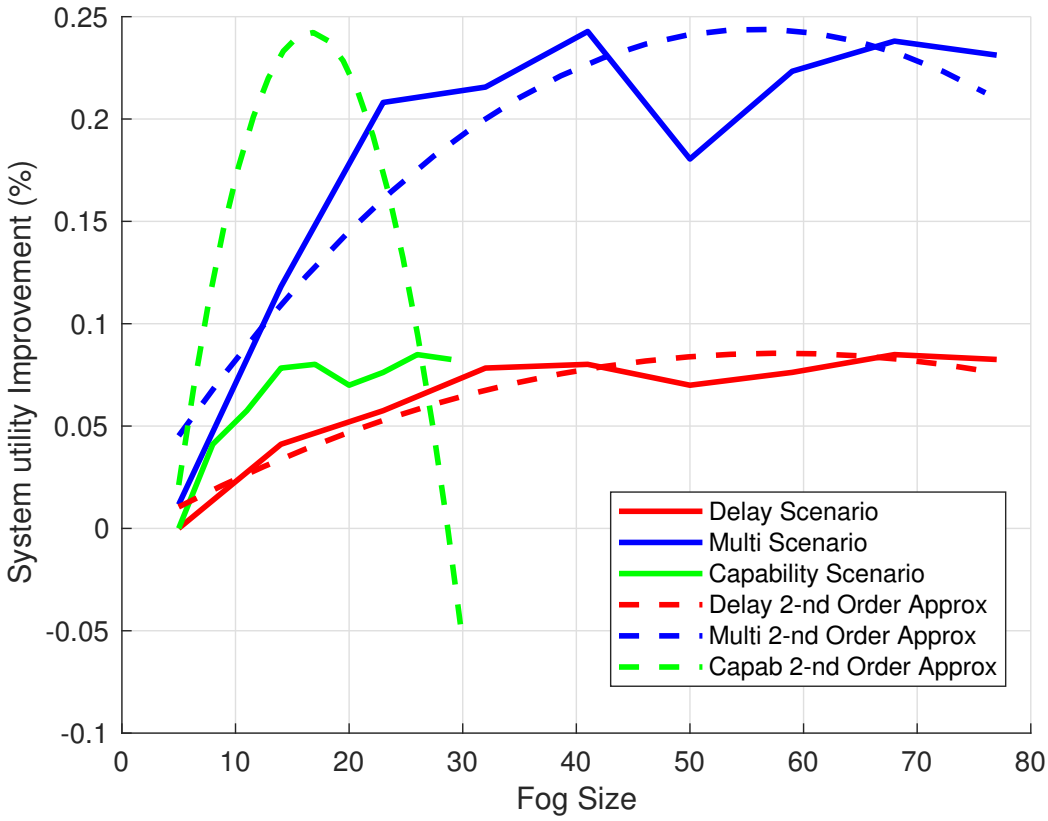

Fig. 6.22 The effect of the Fog Size on Outcomes

analysed fast and results in the upcoming clusters that are based on better weights to get more time and have larger sizes. Based on these, the *minPts* sizes are defined as in Table 6.4

Table 6.4 minPts Parameter Selection

	Delay Scenario	Multi-Parameter Scenario	Capability Scenario
Desired Cls Size	58	55	17
minPts	18	15	5

6.6 Performance Analysis

The scaling tests are designed to show how the proposed methods work in small, middle and large-scale deployment scenarios. The proposed Sampling and Initial Weights training methods are compared to the Modified GA method, the network analysis clustering method (distance clustering) and a random clustering and allocation method to verify how they

achieve results and in what time. The results are plotted in logarithmic time scale and the comparative utility is the ratio of the methods result compared to the best result of the system. If an attempt fails, the method is re-run 3 times to ensure that the failure is a characteristic of the system and not a marginal case, considering that if out of three attempts did not succeed the random starting points or direction is not to blame.

6.6.1 Small-Scale Tests

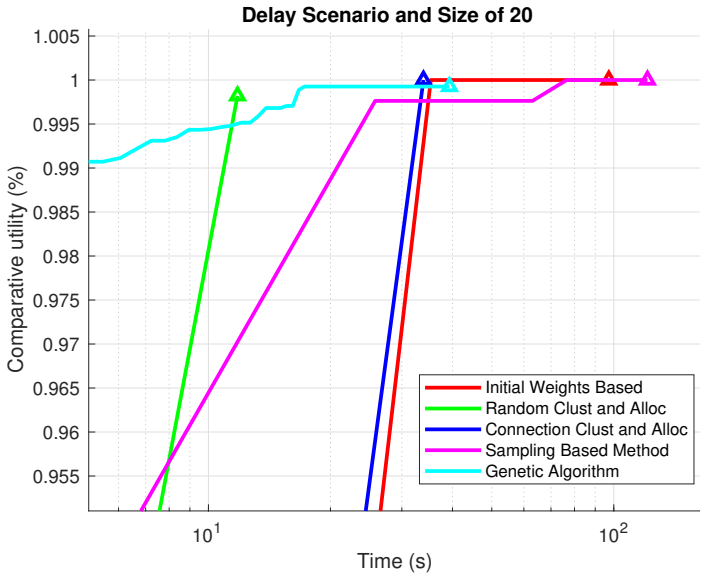

Fig. 6.23 Small scale Delay Scenario Performance test

The small-scale tests are shown in Fig. 6.23 for the Delay Scenario, Fig. 6.24 for the Multi-Parameter scenario and Fig. 6.25 for the Capability Scenario. The small scale test for the Capability scenario has fewer apps as the increased difficulty of this scenario makes differences visible at smaller scales. In these tests, if the results of a test are at 0 or are not visible, it means that there was not valid solution. This is true for the large test set as well.

From the results of the tests, it is obvious that as the difficulty of the scenarios increases the effectiveness of the simple methods and their resulting utility is diminished. In these tests, the scale of deployment is low enough, so the methods find similar or the same best results. In the case of the Capability scenario, they find the same results. In Fig. 6.25 the methods are tweaked to find clusters larger than the size of the Fog deployment so all methods find the same cluster which is, in fact, the generated Fog environment. The GA is the first to finish as it does not need to attempt clustering or resource allocation.

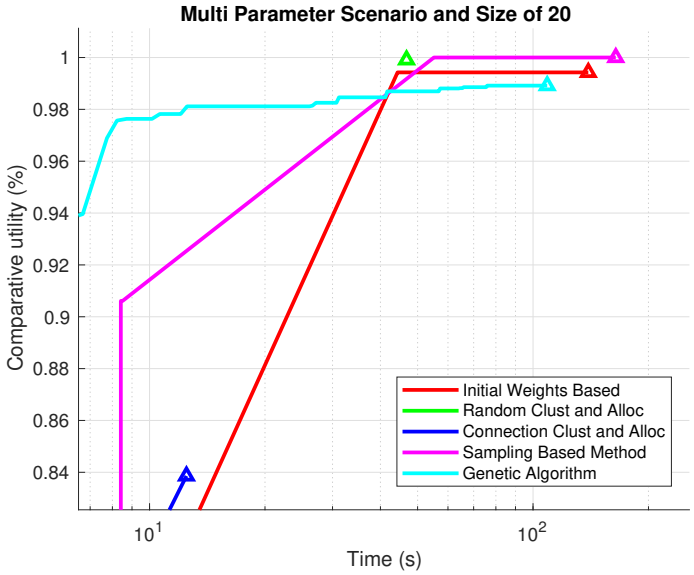

Fig. 6.24 Small scale Multi-Parameter Scenario Performance test

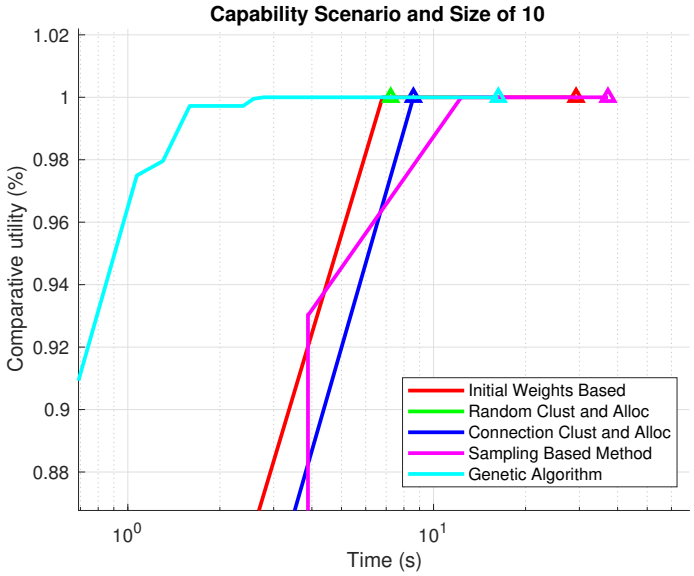

Fig. 6.25 Small scale Capability Scenario Performance test

From the execution times, the benefits of the simple methods are visible as they terminate much faster than the others. At these scales, the Global GA finds solutions a lot earlier than the proposed methods, and the quality of these solutions at the same time is similar or equal.

Another element that can be seen in these tests is the reduced iteration count for the two proposed methods, which can be attributed to the reduced size and lack of room for improvement, as it can be seen in the GA scaling tests from Fig. 6.22.

6.6.2 Medium-Scale Tests

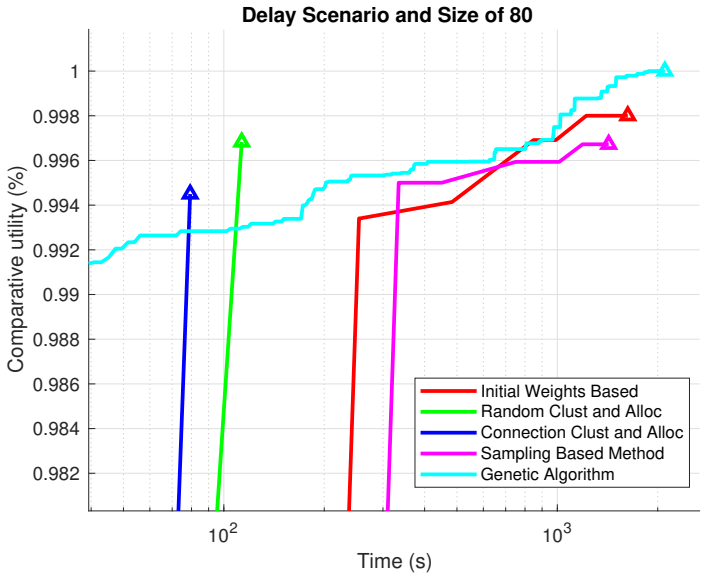

Fig. 6.26 Medium scale Delay Scenario Performance test

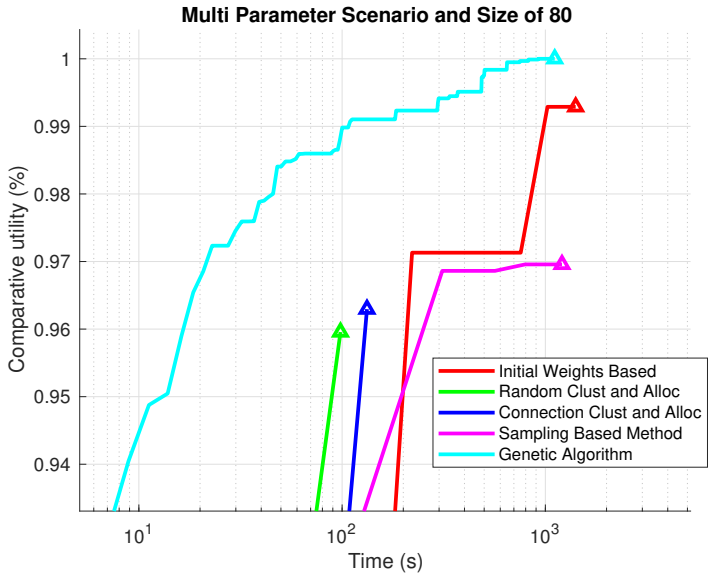

Fig. 6.27 Medium scale Multi-Parameter Scenario Performance test

The medium scale tests are shown in Fig. 6.26 for the Delay Scenario, Fig. 6.27 for the Multi-Parameter scenario and Fig. 6.28 for the Capability Scenario. These tests are designed to show how the methods perform in an environment that is more suited to their purpose.

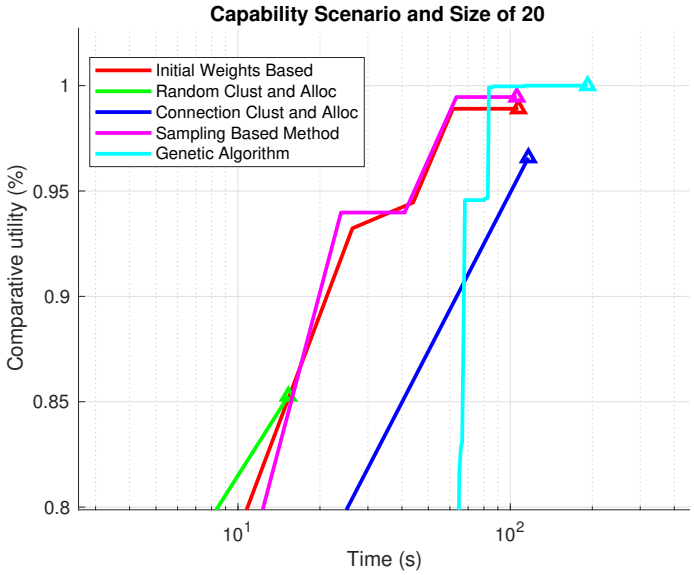

Fig. 6.28 Medium scale Capability Scenario Performance test

Given enough room for improvement and small enough scale the GA method outperforms the other methods in the first two types in Fig. 6.26,6.27, but fails to find a solution in Fig. 6.28. Here, the proposed methods outperform the simple ones and come close to the GA. This supports the need for larger clusters where enough room for improvement is given. In this case, the proposed methods run more iterations as in the previous tests, but still do not come close to the given limit of 10.

When looking at the execution time of the first two tests, it can be said that the proposed methods find solution having a worse utility than the GA as at this scale the Global GA method is close to it's ideal running conditions.

6.6.3 Large-Scale Tests

The large-scale tests are shown in Fig. 6.29 for the Delay Scenario, Fig. 6.30 for the Multi-Parameter scenario and Fig. 6.31 for the Capability Scenario. These tests are designed to show how the methods perform in a large scale environment for which they were designed.

In these tests, the benefits of the proposed methods become more prominent as they surpass the GA method in both execution time for a certain Utility and the best utility as well in the Delay scenario. For the Multi-Parameter Scenario, the Initial weights based method performs worse than the GA which shows the importance of finding good starting weights for the system. The Capability scenario shows that at large scales most methods failed to find a solution with only the sampling method being successful.

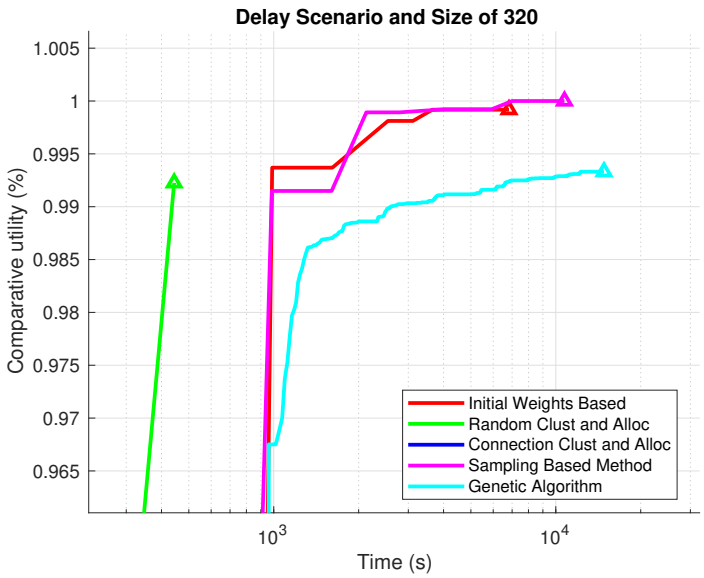

Fig. 6.29 Large scale Delay Scenario Performance test

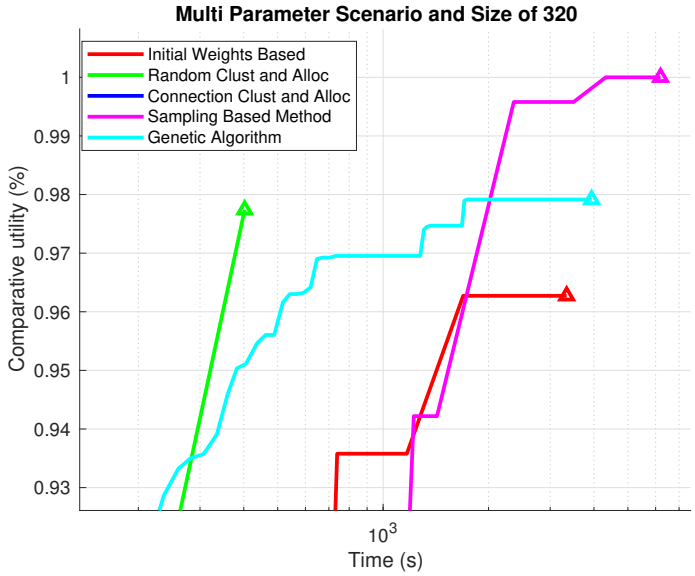

Fig. 6.30 Large scale Multi-Parameter Scenario Performance test

The Connection Clustering and allocation method fails all scenarios as the clusters it finds and their allocations over-fragment the resources which makes finding solutions more difficult. The random Clustering and Allocation works well, as it is done in a balanced way with even and test-based optimal cluster sizes and equal gateway distribution. This is a very crude and simple allocation method but as it can be seen from the tests it provides a decent

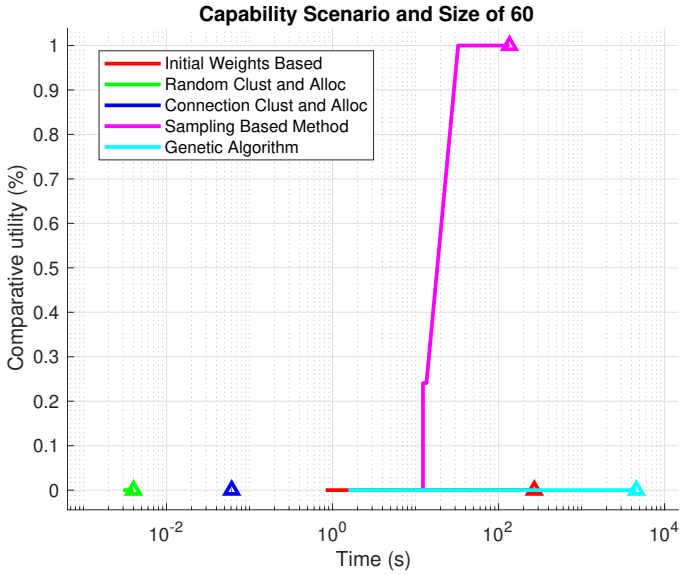

Fig. 6.31 Large scale Capability Scenario Performance test

solution in the first two scenarios comparable to the global GA method and in a lot faster time.

6.6.4 Conclusions

When evaluating the performance tests, the scalability analysis of the GA method from Fig. 6.22 needs to be considered as well. In this test, the GA has its best performance at 58 for the Delay, 55 for the Multi-Parameter and at 16 for the Capability Scenario. What this means for these tests is that by breaking up a Fog of Size 80 or under results in a situation where there is not enough room for the GA to get good results inside the clusters, which explains why the global version works best. Reaching the higher scales improves the situation, but there is still only an average of 7 resulting clusters. Even so, the advantages of these methods are present, as the sampling method outperforms the others and GA in both execution time and best results attained.

6.7 Scalability Analysis

The scalability tests are designed to show an overview of how the methods perform based on the scale of the system, providing less detail than the performance tests on how a certain solution is reached but giving an overview of the results. This set of tests is run by generating 5 Fog deployments of a certain size and scenario and then running all the methods on this

test to verify how well they perform. If a test fails its utility is considered as 0, and if all tests fail a new Fog is generated. As the execution time of the system increases exponentially with the size, this is plotted in logarithmic scale while the utility improvements are plotted as opposed to the best results of the system.

6.7.1 Delay Scenario

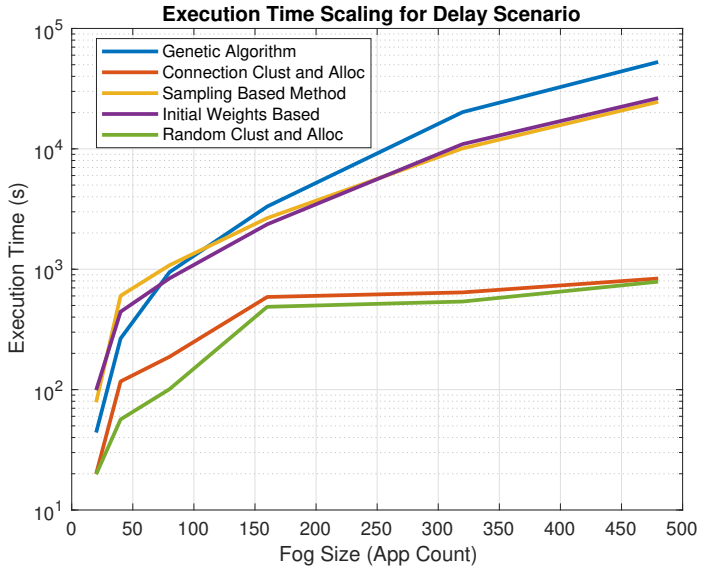

Fig. 6.32 Delay Scenario Execution-Time Scalability test

The delay scenario is designed as a reference situation, as it is the most common objective of most analysed optimisation attempt. The execution time scaling for this can be seen in Fig. 6.32 and the utility scaling can be seen in Fig. 6.33.

The time scaling test shows that the simple random and distance based methods are by far the fastest of the presented ones and that their slopes are smaller as well. When considering the more time-consuming methods it can be seen that with deployment sizes below 120, the GA method performs faster than the proposed methods. For sizes above this, a clear distancing between the two can be seen which would be accelerated at higher scales.

The Utility scaling for this scenario shows how at lower scales, under 200 most methods have comparable results with little differences while at higher scales the problem of finding a valid deployment comes into play. The beginning of GA limitations can be seen here, as it fails to find valid solutions for any of the tests above the 320 apps mark. It is also worth noting that the Random and Initial weights methods perform in similar fashion while the

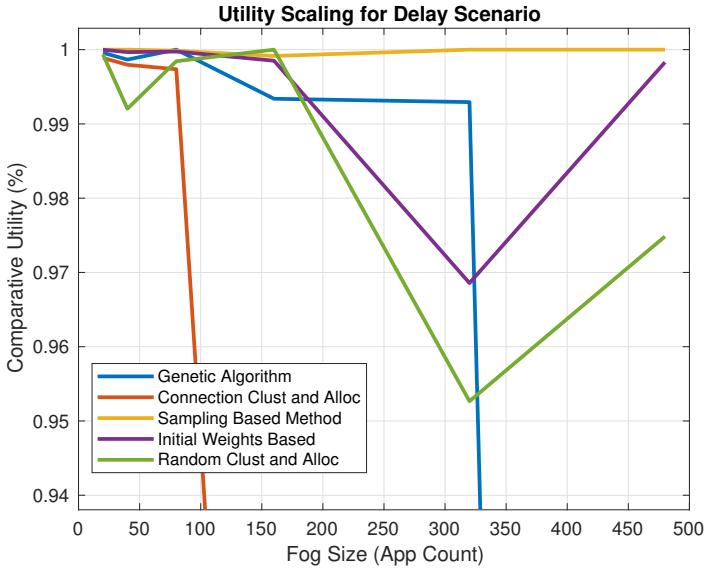

Fig. 6.33 Delay Scenario Utility Scalability test

sampling methods are consistently the best, showing the importance of finding good starting weights and the power of clustering as well.

6.7.2 Multi-Parameter Scenario

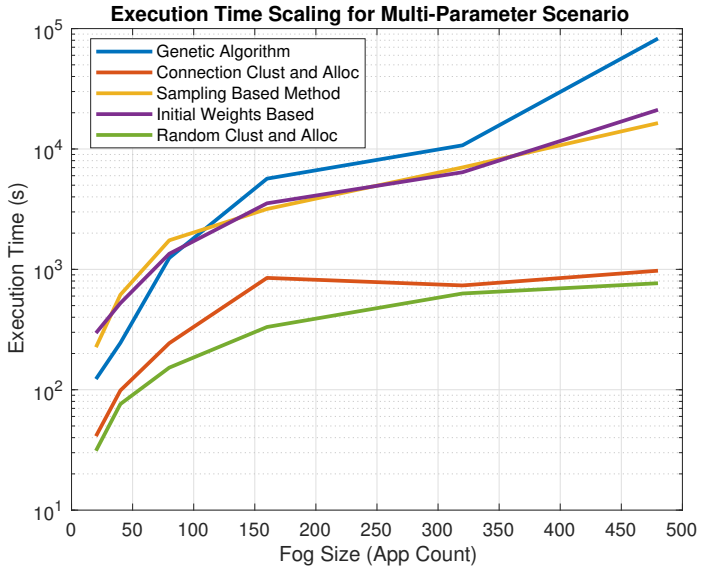

Fig. 6.34 Multi-Parameter Scenario Execution-Time Scalability test

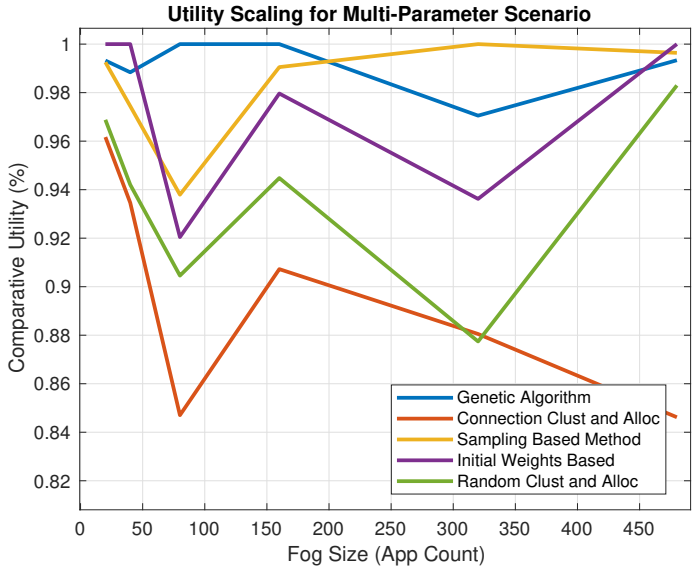

Fig. 6.35 Multi-Parameter Scenario Utility Scalability test

The multi-parameter scenario is designed to be a challenge for the clustering methods, as it has varying weights and changing soft constraints for apps which makes it difficult for the methods to find a valid direction for clustering or resource allocation. The execution time scaling for this can be seen in Fig. 6.34 and the utility scaling can be seen in Fig. 6.35.

The time scaling tests show a similar trend as in the previous subsection, with small divergences and a larger gap between the GA and the other methods as well as a steeper incline and increased processing time for all the methods. This shows that the more difficult the optimisation problem is, the more resources are required to solve it.

Form the utility tests, a number of conclusions can be drawn. One of these supports the previous tests, where the GA outperforms the clustering methods on lower scales where it is able to find better solutions, but at sizes larger than 300, the clustering methods get better results.

6.7.3 Capability Scenario

The capability scenario is designed to test the limits of the methods and reinforce the findings of the performance tests. It is designed to provide a scenario where usually first-fit style methods are used. The execution time scaling for this can be seen in Fig. 6.36 and the utility scaling can be seen in Fig. 6.37.

The time scaling tests for the capability scenario shows the intensification of the trend that could be seen in the previous tests where the GA method had increased its execution

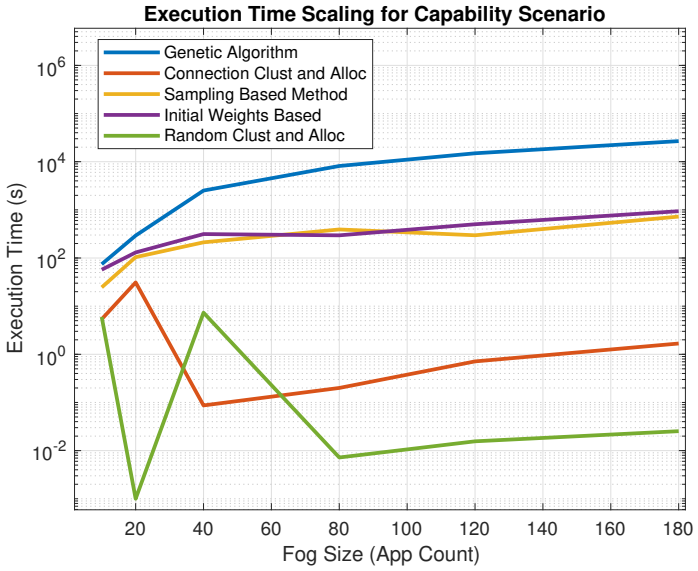

Fig. 6.36 Capability Scenario Execution-Time Scalability test

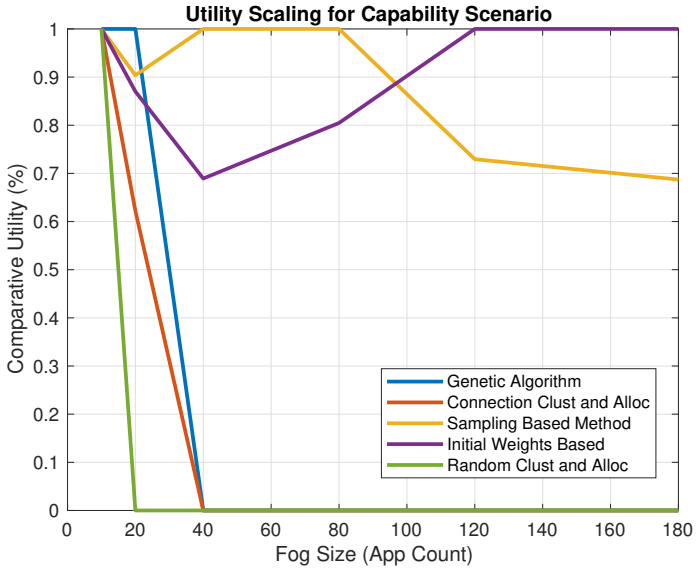

Fig. 6.37 Capability Scenario Utility Scalability test

time with the increase of the deployment scenarios' difficulty. This can also be seen here with the sampling and initial weights methods performing similarly while the GA method reaching execution times that were 10 times higher. This is mostly due to the difficulty of finding valid solutions which is the main characteristic of this scenario.

From the scaling tests the results of the performance evaluation in Fig. 6.31 are confirmed as only the two weights based clustering methods are able to find solutions with the initial

weights based methods overtaking the sampling one. This can be explained with the over-confidence in certain weights caused by the initial sampling, which is smaller cases allows for faster results but in larger scenarios may cause a fast convergence. The large differences between results also support the difficulty of finding even one valid solution.

6.7.4 Conclusions

When considering the execution time scalability, it can be concluded that the proposed methods provide a way to reduce the execution time of the GA method and reduce the slope at which it increases with scale. It is worth noting that this difference can mostly be observed with Fog sizes larger than 150 applications. This difference would be more significant if the Local allocation GA would be distributed so the different clusters could be evaluated separately. This, however, would result in comparing a single-thread process with a multi-thread one.

The utility improvements of the system follow a similar pattern as the execution time wherein small scales, under 300 apps, the GA method outperforms the proposed ones but as the scale of the system increases, the benefit of clustering becomes more evident. From these tests a limitation of the weighted clustering methods can be observed, as well as they perform poorly in the multi-parameter scenario where there is no clear direction to what makes a good utility as in the case of the delay improvement and the Capability scenario.

6.8 Component Evaluation

The component evaluation section is designed to show the characteristics of each method, broken down to its components. In this section, the Clustering components, Resource Allocation and weight Training components will be analysed. This is done to show how each contributes to the overall execution time and resulting utility that can be found in the previous figures.

An overview of the execution times of these components can be seen in Fig. 6.38 where the scaling of the execution times for these components can be seen. In this figure, the execution times are shown in logarithmic scale as the method vary largely in their run-time duration.

From this figure, the Clustering GA only shows the average time for each cluster and not their values combined. From these tests, the exponential increase in the processing time for Clustering can be noted with the other components retaining lower slope gradients.

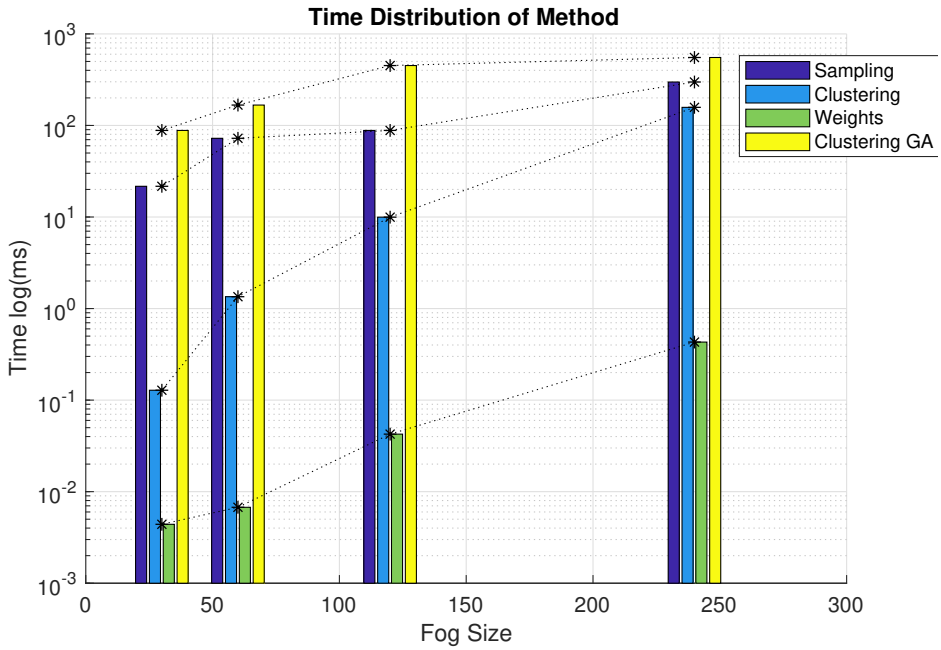

Fig. 6.38 Components Time Distribution

6.8.1 Resource Allocation

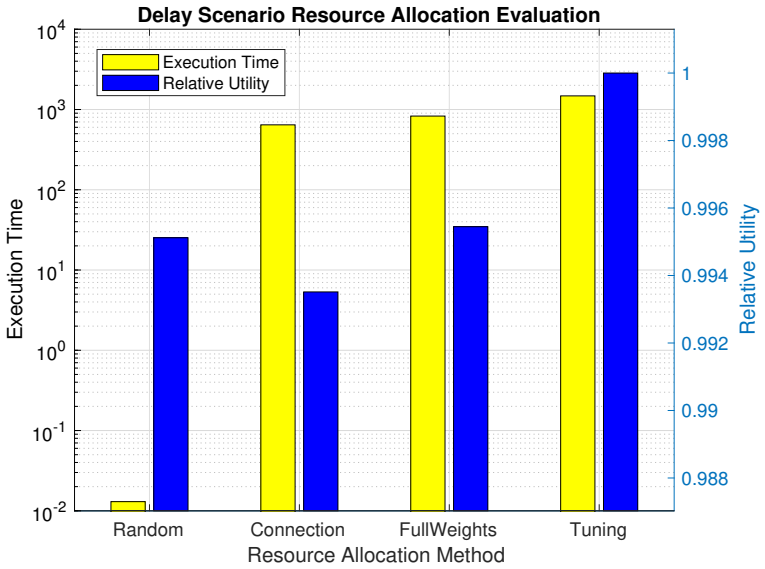

Fig. 6.39 Resource Allocation Comparison Delay Scenario

The resource allocation testing scenario is designed to compare Allocation methods based on their execution time and resulting utility after deployment. For these tests, only the Delay

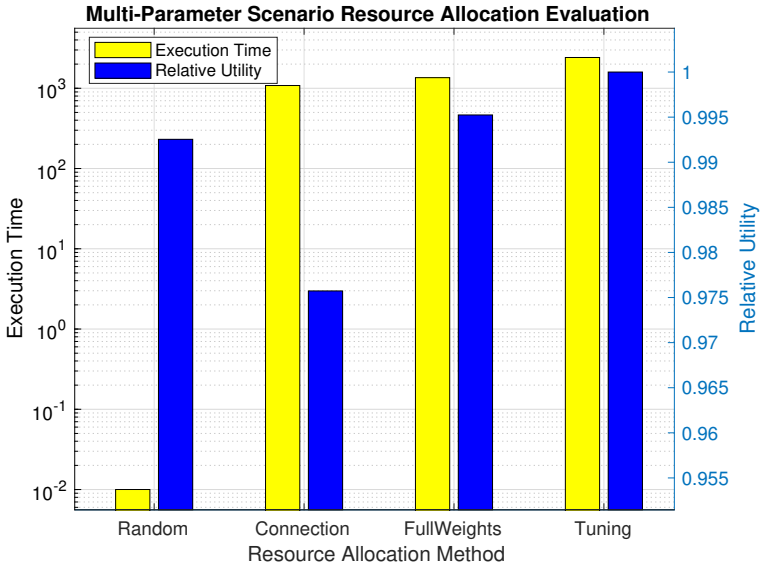

Fig. 6.40 Resource Allocation Comparison Multi-Parameter Scenario

and Multi-Parameter scenarios are considered, as the Capability scenario requires weights tuning to provide a valid solution resulting in only the Tuning based Resource Allocation to have valid results.

These tests are run by generating a Fog environment of size 320 for the Delay and Multi-Parameter scenario after which the Sampling method is run to identify to best weights for the system. After the weights are identified, the Fog is Clustered using the Weighted Clustering method and the tuned weights after which each allocation method is used to allocate gateways to the clusters. Their run-time is recorded and local GA is run to identify the resulting utilities.

In the Delay scenario from Fig. 6.39 , the Random, Connection and Full Weights based methods have comparable result, with the Connection based on being the worst. The Full Weights based method considers all app to gateway parameters equally important which is shown to be almost as good as considering them at random while considering connection parameters as the most important leads to the worse results. The quality of the results based on the tuning weights allocation shows the importance of finding good weights and allocating resources to clusters in this way.

The Multi-Parameter scenario in Fig. 6.40 shows the difficulty of allocating Gateways to clusters that have a high heterogeneity and varying utilities and objectives. Here the quality of results produced by the connection-based allocation worsens, and as more parameters gain importance the quality of the Full weights allocation comes close to that of the Tuned weights, which despite this still provides the best solution.

When considering the execution times of these methods, it's worth noting that the random allocation while not having the best performance in the utility category is by far the fastest method. The execution time for the rest of the methods scale as expected with their complexity.

6.8.2 Clustering

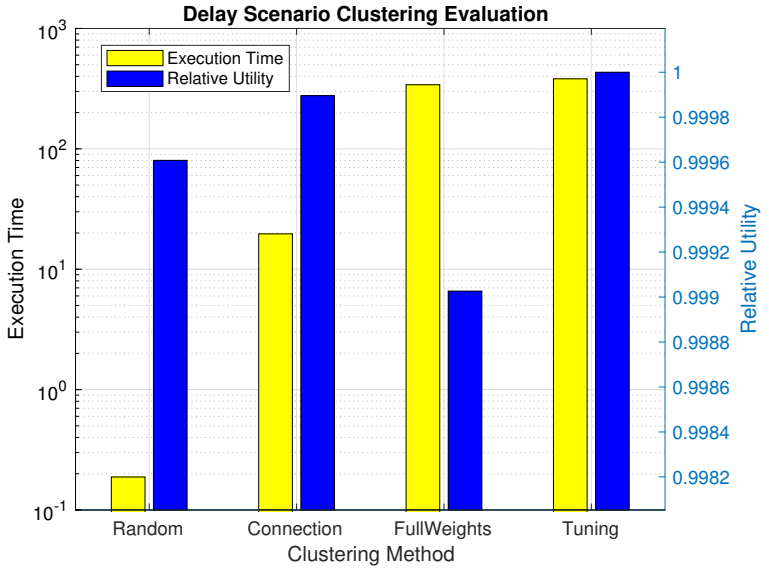

Fig. 6.41 Clustering Comparison Delay Scenario

The Clustering evaluation tests are designed to test and compare the proposed clustering methods to find their run-time characteristics. For these tests, only the Delay and Multi-Parameter scenario are considered as well based on the same rationale as in the previous subsection.

These tests are run by generating a Fog environment of size 320 for the Delay and Multi-Parameter scenario after which the Sampling method is run to identify to best weights for the system. After the weights are identified, the Fog is Clustered using the four methods. Their run-time is recorded and then a weighted allocation method based on the calculated weights is run to allocate gateways to the clusters. Finally, a local GA is run to identify the resulting utilities which are then attributed to the clustering methods.

In the Delay scenario in Fig. 6.41 the advantages of the Connection based clustering are visible as the main reason two application should be deployed together is to reduce the connection delay between the two. This fact can be seen from the tuned weights that are found for the scenario as from the results as well. In these tests the Full Weights and Random

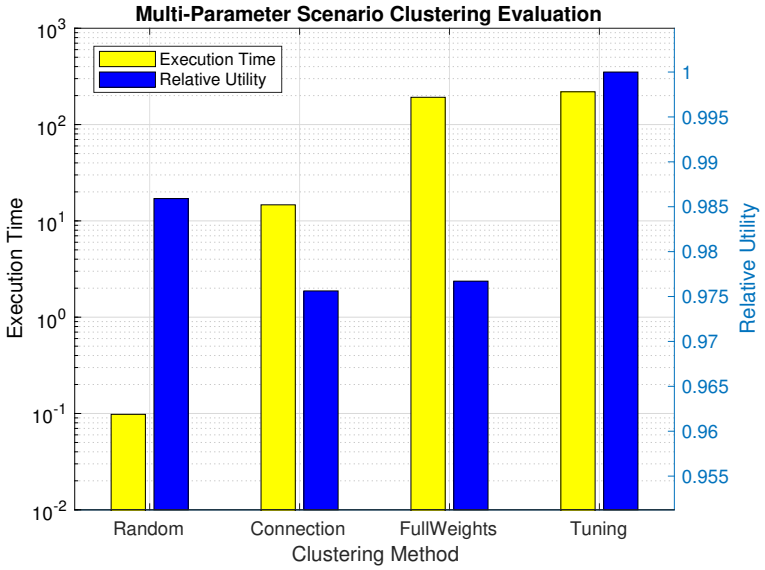

Fig. 6.42 Clustering Comparison Multi-Parameter Scenario

allocation does not perform too well as there is a clear direction for clustering. As in the previous scenarios, the Tuning parameters based method works best.

The Multi-Parameter scenario in Fig. 6.42 has similar results as for the Resource Allocation of the same scenario where the Connection Clustering and Full Weights methods do not perform so well while the Random but fair allocation has close results. In this situation, the Tuning methods perform best as well.

The execution time for this set of tests resembles that of the allocation scenario with the Random Clustering outperforming the Connection and Full weights based allocation at utility and overall having the lowest execution times. The rest of the methods performed as expected with the Full Weights and Tuning Weights ones being the slowest.

6.8.3 Weights Tuning

The Weights Tuning tests are designed to show how the training algorithms works, how it finds new weights and how the under/over-fitting components works. The tests were performed on all the scenarios, as they propose different challenges and the performance of the methods varies for each scenario. The tests were conducted at Fog Sizes of 320 for the Delay and Multi-Parameter Scenarios and with a size of 120 for the Capability Scenario. To run the tests an initial Fog Environment is generated and both methods are deployed in the same environment. With the difference in desired cluster sizes, all scenarios resulted in the methods generating cluster counts ranging from 7 to 11.

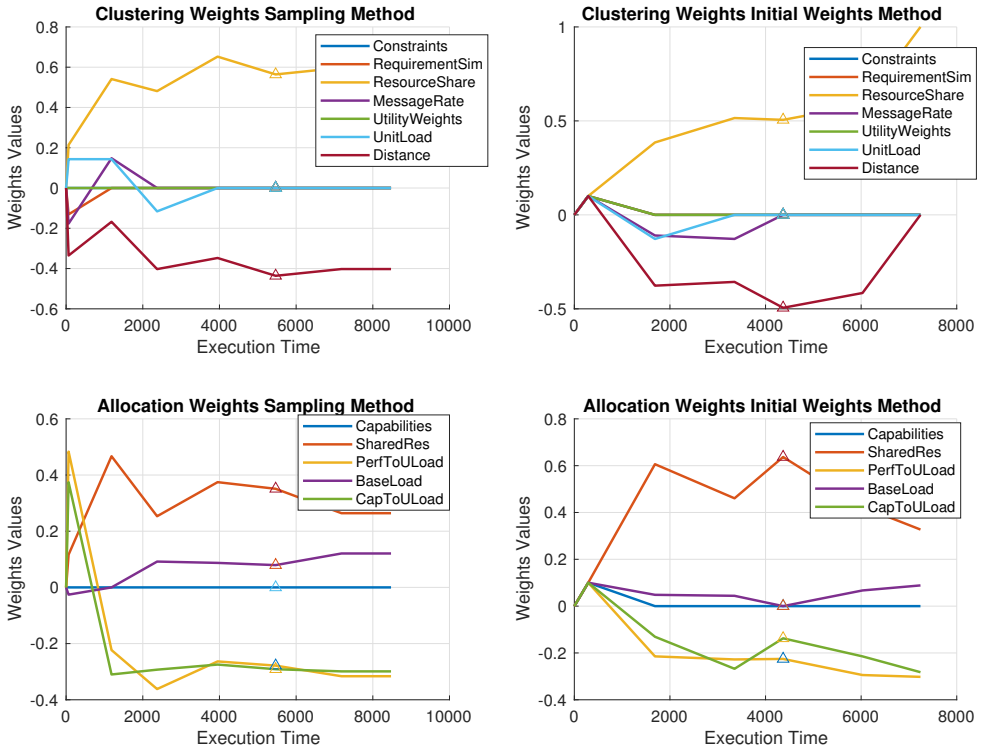

Fig. 6.43 Weights Tuning Evaluation for the Delay Scenario

The resulting outputs are the weights for each property at different iterations showing at what time those iterations were found. The weights that resulted in the best Utilities are marked with triangles of the same colour. The three scenarios can be seen in Fig. 6.43 for the delay scenario, Fig. 6.44 for the Multi-Capability scenario and Fig. 6.45 for the Capability Scenario.

Considering the Delay Scenario tests from Fig. 6.43, the impact of the initial sampling can be seen at the lower end of tests where there is a spike for certain parameters, after which when the whole system is deployed the confidence in these parameters drops. This phenomenon can be seen in some of the later tests as well as in due to the relatively small size of the sampling test, which is set to 10% of the total size. This spike or rough initial values perform better for the stability of the system, as opposed to the Initial Weights tests where these values fluctuate more, due to the initial weights set where all parameters are equal. For this delay scenario, the Initial weights tests took longer to complete as well as the fluctuations reduced the change of stagnation. This is sometimes beneficial and results in better utilities. The best utility for the scenario of 824.06 was found with the App Weights $\{ResourceShare = 0.505, Distance = -0.494\}$ and Gateway Weights

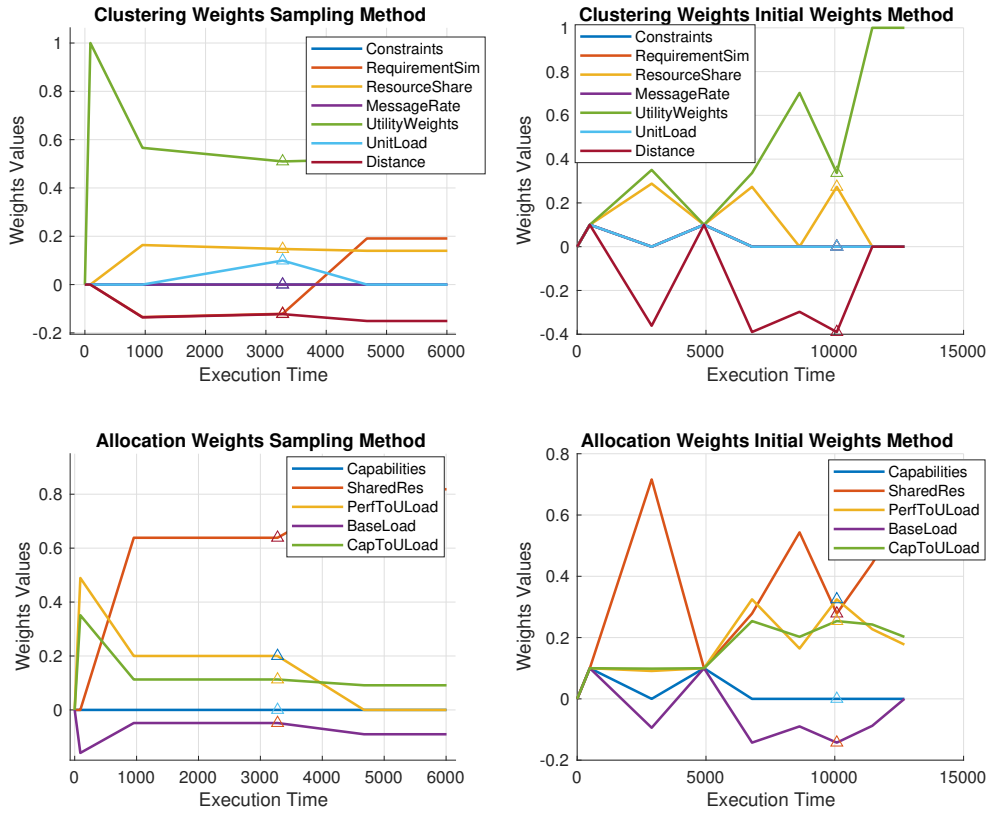

Fig. 6.44 Weights Tuning Evaluation for the Multi-Parameter Scenario

of $\{SharedRes = 0.637, PerfToULoad = -0.225, CapToULoad = -0.137\}$ by the Initial Weights scenarios at 4560.59 seconds showing the advantage of the confidence adjustments. The solution found by the sampling method was just 1 point off at 823.21 and was found at time 5421.72.

The results of the Multi-Parameter tests can be seen in Fig. 6.44 where the initial spikes of the sampling method are once again visible. The instability of the initial weights approach can be seen as well with large variation in parameters and with even a reset of these at 4923.41, where the weights did not result in any valid deployments and the initial weights were the only valid one s known. This instability resulted in a worse result for the initial weights approach. The best utility for the scenario of 276.70 was found with the App Weights $\{UtilityWeights = 0.509, ResourceShare = 0.147, Constraints = 0.099, Distance = -0.121, RequirementSim = -0.122\}$ and Gateway Weights of $\{SharedRes = 0.638, PerfToULoad = 0.201, CapToULoad = -0.112, BaseLoad = -0.048\}$ by the Sampling Weights scenarios at 3277.234 seconds showing the advantage of sampling for the conver-

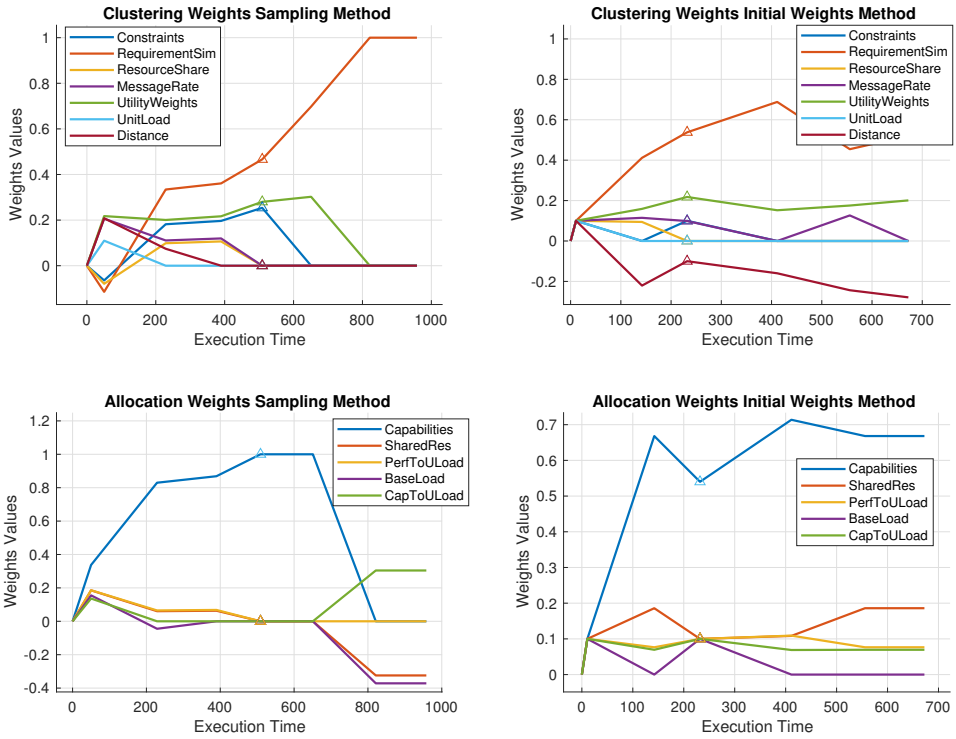

Fig. 6.45 Weights Tuning Evaluation for the Capability Scenario

gence of these methods. The solution found by the initial weights method was just 266.21 and was found at time 10078.12

The Capability Scenario in Fig. 6.45 shows a case where the initial sampling results were counterproductive and sent the method in the wrong direction resulting in more than 200 seconds of added time in finding similar weights as the Initial weights method did. This scenario has a clear set of goals as well, where the capabilities to requirements fitting are key. This makes it a more suitable scenario for the method as opposed to the Multi-parameter scenario that has numerous directions of interest. The best utility for the scenario of 104.54 was found with the App Weights $\{UtilityWeights = 0.280, Constraints = 0.253, RequirementSim = 0.466\}$ and Gateway Weights of $\{Capabilities = 1.0\}$ by the Sampling Weights scenarios at 516.97 seconds showing the advantage of sampling for the convergence of these methods. The solution found by the initial weights method was 103.42 and was found at time 231.91.

6.8.4 Conclusions

When considering the execution time of the methods through the varying scenarios and component test, several conclusions can be drawn. In the overall picture, the Resource Allocation and Clustering still require a fraction of the time of the local GA deployments, but as the size of the Fog increases so does the impact of these methods. The execution time of the sampling method is comparatively low as well, and it does not suffer from the scaling issues the Resource Allocation and Clustering do.

From the utility comparison of the individual components, the main conclusion is that while the Random allocations and clustering are much faster than the other methods, determining the right weights for a system, clustering and allocating resources based on these provides the best solution in most cases. In the case of the Capability scenario, this is the only one that can reliably provide solutions. From these tests, the conclusion can be made that using the wrong weights for a system can perform worse than random deployments which highlights the importance of the identification and tuning of these weights.

From the weights tests section, some of the benefits and drawbacks of using sampling become apparent, as having some initial weights for deployment can result in finding a solution faster as in having a faster convergence, but this might result in overconfidence over the results of the sampling. The steps increase in the tests with the increase in the Fog size is also notable. This can be seen in both the performance evaluations and in the differences between the Delay and Multi-Parameter tuning and Capability tuning tests.

The initial weights scenario as it allows all parameters to be considered, while sometimes finding better solutions, most of the times struggles with determining the right weights and in the case of the Capabilities scenario this usually means that no, or limited valid solutions are found. These tests also support the need for adequate sample selection and the need to learn from failed attempts.

Chapter 7

Conclusions and Future Work

7.1 Results Overview

In this chapter, an overview of the works will be presented alongside the initial requirements, objectives and the analysis of the outcomes. Each component of the framework will be scrutinised to identify the novelties, contribution to the state of the art and the areas where it may fall short of the requirements or the downsides of certain approaches.

When looking at the framework as a whole it provides a complete image of platform support for application hosting migration and systems setup on top of which the evaluation and improvement of deployments based on the proposed model through the optimisation methods are possible.

There are some limitations however as the proposed scenario looks at a Shared Environment based Gateway and thus the model and method are tailored for such a scenario. VM and Container based solutions would offer simpler Models and may require simpler methods to solve with the underlying problem being solvable in polynomial time. Furthermore, the scaling and testing data generation is based on the WWW scaling model that was identified in the use cases. In situations where this is not present the clustering methods would be less effective.

7.1.1 Platform Review

The proposed Fog and IoT platform allows applications to be deployed closer to the network edge and migrated to the Cloud based on the users' requirements. Using the OSGI gateway for application deployment allows life-cycle management of applications as well as the deployment of a set of applications as they can work together in a Micro-service environment.

Furthermore, this solution allows parts of the applications to be migrated to the Cloud where the more processor intensive tasks might be performed.

Compared to similar research in the field, this platform, through dynamic abstraction, allows for a protocol agnostic application environment, as well as a modular deployment of applications to the gateway. The platform also provides a solution for the increased horizontal integration of devices by allowing multiple tenant connections to be configured from the gateway that may use resources available from different providers. It also provides for speedy creation of test environments and the option of migrating between Cloud and gateways on the region depending on processing needs. Furthermore, the gateway makes steps towards a better horizontal integration by allowing the connection through different drivers to local and Cloud resources while allowing different application environments and device connections. In an industrial environment, this would allow for faster time to market, a more dynamic production environment, faster software upgrades and easier testing.

The limitations of the presented gateway lie in the added overhead caused by having to translate messages from one driver/protocol to the system's protocol as well as in the security and group reliability issues caused by the shared environment. The overhead caused by these components can be seen in the models for the routing and message loads. These are relatively small compared to the characteristic ping or networking delays but they are present and need to be acknowledged. Furthermore, the drivers and brokers may cause bottlenecks on the system as their implementation might not be designed for high data-rates but this is dependent on the developers and is not a characteristic of the framework. The security and interdependence issues arise from applications from different providers being deployed in the same environment, making their interaction possible. This can cause problems if an application overloads the system or attempt snooping on plug-in data mining activities.

7.1.2 Model Review

This model provides a way of measuring and estimating the run-time parameters and migration benefits of applications in these shared environment systems. The experimental load model description derived from measuring run-time parameters over physical systems has been developed and used to represent the gateway and application loads, which provide a more realistic estimation than theoretical ones presented in other papers. The experimental results have shown that the system has an overall accuracy of over 91%.

The assignment problem that results from attempting to minimise the proposed utilities is an NP hard placement problem with interdependent parameters that has proven to be a challenging one for both heuristic and deterministic methods, neither being able to provide the best results in all cases.

Some of the drawbacks of the proposed model are related to its narrow parameter focus, its generality and the resulting optimisation problems hardness. The presented model looks at estimating the load of the system and applications and derives the delays and the reliability from these considering constraint and weights as well. The later can be considered as SLA or QoS Constraints and requirements but addressing these directly is not done. Furthermore, recent trends show that Energy use is an increasing factor which is not considered in this work, as replication and zero points of failure reductions either. Due to the testing environment, the generality of the model is affected as it would need simplification to work with VMs and Container, but it would work with other Containerised Python or Ruby based shared systems. The final problem with the proposed solution is the complexity of the model. Solving the system for this model results in an NP-hard problem, so sacrificing some of the accuracy of the model for the sake of an easier allocation problem might be worth considering.

7.1.3 Deployment Method Review

The proposed global optimisation method attempts to solve the problem of the exponentially increasing search-space in case of the Application to Gateway allocation problem in the presented Fog Systems. The solution for this is to reduce or to split this search-space in such a way that as little information or possible good solutions are lost. This method attempts to do this by forming clusters and assigning resources to these in effect choosing solution regions that are then optimised locally. The components of the method aim to find ways of grouping applications so that these groups have the highest possible utility. This is done by looking at their properties and how they relate to each-other and to the gateways and finding those properties whose similarity makes applications deployed together result in a higher utility.

This solution proves to be very effective both for scalability and for improved results, given a certain size of Fog system. This is partially thank to the lack of initial assumptions made about the system and also about allowing the methods to figure out the interesting variables. This generality has its toll however as for the simple instances the connection clustering methods find similar solutions but in less time. Based on the validation tests it can be concluded that the optimisation method works best in large-scale environments that have a complex set of requirements and utilities, where the solution for attaining these would be difficult to find by a human.

The drawbacks of the system are liked to the core approach of the design where everything is based on greedy or quick-sort style algorithms where the best solution is not as important as a good solution. This can lead to sub-optimal results, but due to the number of iterations and the size of some problems still results in large processing times. Furthermore, the training

and sampling algorithms while sometimes finding new and better solutions through direction changes, in most cases fail to do so. All of these are addressed by the generality to time complexity figure as well as the scalability analysis in the validation section that shows where concessions are made between the method variations, which would support the decision on which to use for a system.

7.2 Answer to Research Questions

When considering the research questions and how these were answered, the niche questions are shown first and what was found. Based on these the big questions are answered and their completeness analysed.

Niche Questions

- **What are the requirements and characteristics of future Industry 4.0 Gateways and how can these be translated into protocols and systems?**

To answer this question, a number of components have been suggested such as the virtualisation of devices and the translation of messages to allow for more interoperability and horizontal integration as opposed to vertical integration. A more decentralised system is suggested as well through the gateway platform. These directions were then translated into a framework and implemented and tested in the physical environment.

- **How can changes in the model be analysed and estimated using the run-time parameters and connections of the applications and gateways?**

A number of testing use-cases were deployed and their applications were migrated between Fog nodes and their Cloud-based counterparts. The effects of the migration were and key parameters noted. A literature review of the parameters of interest and the testing parameters led to the formulation of the model. This model based its estimations on the connections and linked nature of applications.

- **What are the challenges of application deployment in Fog systems and what methods can be proposed to diminish their effects?**

Use-Cases were analysed to see how Fog systems scale and what is their typical structure. This data was used to determine the hardness of their deployment and through this analysis methods were proposed that reduced the search-space to improve performance indices.

Main Question

- **How can large application systems deployments be analysed and improved in highly heterogeneous Fog environments?**

To answer this question, first, a platform was designed that can house future Industrial applications. Based on this platform an application and gateway model was formulated that allows the real-time and offline analysis of these systems. Finally, a global optimisation method is proposed that aims to improve system health and allow for the deployment of large highly heterogeneous systems.

7.3 Future Work and Directions

There are a number of future directions or research interests that can be expressed based on this thesis. These can be related to the missing components and the limitations of this work, but also to niche problems or opportunities that were not addressed.

• When considering the Platform a number of improvements can be made. A number of papers have suggested a platform model that looks at the main brokers, translators and dongles these platforms need to encompass a good portion of the existing technologies. Implementing and testing these would reveal the true possibilities and perspectives of such gateways.

• The deployment of multiple application containers or VMs on a system might aid in the security and interdependence issue of Shared systems and would increase their generality as you could deploy Java Python and other apps on one system, reducing platform lock-in. This would result in more complication optimisation scenarios as in the 4th type but could possibly increase the resulting system utility.

• The existing model can be generalised even further by considering the characteristic behaviour of a number of application types as some may be distributable running multiple threads as well as looking at the reliability differences between gateways and the Cloud and putting these in a formal model.

• The proposed model can be extended to look at QoS improvements, Billing and Energy Consumption reduction.

• While correlation calculation was used to determine weights and the clustering methods were based on these machine learning methods could be applied to these systems to verify if this allocation or clustering could be identified through these as well.

• The proposed method is a global optimisation approach to application deployment. Load Balancing style approaches can be proposed that are based on the existing clusters

where a mechanism could add new peers to existing clusters, create new clusters and then locally redistribute resources to match this change.

• Agents based systems could be deployed to both create clusters and to allow these collaborative groups to then compete for resources which are the gateways. Clustering the applications is a good start for defining the game, but more work needs to be done in this direction.

References

Aazam, Mohammad and Eui-Nam Huh (2014). "Fog computing and smart gateway based communication for cloud of things". In: *Future Internet of Things and Cloud (FiCloud), 2014 International Conference on*. IEEE, pp. 464–470.

— (2015). "Fog computing micro datacenter based dynamic resource estimation and pricing model for IoT". In: *Advanced Information Networking and Applications (AINA), 2015 IEEE 29th International Conference on*. IEEE, pp. 687–694.

Aazam, Mohammad, Imran Khan, et al. (2014). "Cloud of Things: Integrating Internet of Things and cloud computing and the issues involved". In: *Proceedings of 2014 11th International Bhurban Conference on Applied Sciences and Technology, IBCAST 2014*, pp. 414–419. DOI: 10.1109/IBCAST.2014.6778179.

Aggarwal, Deepak kumar and Rajni Aron (2017). "IoT based Platform as a Service for Provisioning of Concurrent Applications". In: arXiv: 1711.10685. URL: http://arxiv.org/abs/1711.10685.

Aibinu, A. M. et al. (2016). "A novel Clustering based Genetic Algorithm for route optimization". In: *Engineering Science and Technology, an International Journal* 19.4, pp. 2022–2034. ISSN: 22150986. DOI: 10.1016/j.jestch.2016.08.003. URL: http://dx.doi.org/10.1016/j.jestch.2016.08.003.

Alliance, OSGi (2003). *Osgi service platform, release 3*. IOS Press, Inc.

Ankerst, Mihael et al. (1999). "OPTICS: ordering points to identify the clustering structure". In: *ACM Sigmod record*. Vol. 28. 2. ACM, pp. 49–60.

Azeez, Afkham et al. (2010). "Multi-tenant SOA middleware for cloud computing". In: *Cloud computing (cloud), 2010 ieee 3rd international conference on*. IEEE, pp. 458–465.

Baccarelli, Enzo et al. (2016). "Energy-efficient dynamic traffic offloading and reconfiguration of networked data centers for big data stream mobile computing: review, challenges, and a case study". In: *IEEE Network* 30.2, pp. 54–61.

Barreto, L., A. Amaral, and T. Pereira (2017). "Industry 4.0 implications in logistics: an overview". In: *Procedia Manufacturing* 13. Manufacturing Engineering Society International Conference 2017, MESIC 2017, 28-30 June 2017, Vigo (Pontevedra), Spain, pp. 1245–1252. ISSN: 2351-9789. DOI: https://doi.org/10.1016/j.promfg.2017.09.045. URL: http://www.sciencedirect.com/science/article/pii/S2351978917306807.

Bauer, Matthias, Gunther May, and Vivek Jain (2014). "A wireless gateway approach enabling industrial real-time communication on the field level of factory automation". In: *Emerging Technology and Factory Automation (ETFA), 2014 IEEE*. IEEE, pp. 1–8.

Bellavista, Paolo and Alessandro Zanni (2017). "Feasibility of fog computing deployment based on docker containerization over raspberrypi". In: *Proceedings of the 18th International Conference on Distributed Computing and Networking*. ACM, p. 16.

Beran, Peter Paul, Elisabeth Vinek, and Erich Schikuta (2011). "A cloud-based framework for QoS-aware service selection optimization". In: *Proceedings of the 13th International*

Conference on Information Integration and Web-based Applications and Services - iiWAS '11. New York, New York, USA: ACM Press, p. 284. ISBN: 9781450307840. DOI: 10.1145/2095536.2095584. URL: http://dl.acm.org/citation.cfm?doid=2095536.2095584.

Bhondekar, Amol P et al. (2009). "Genetic algorithm based node placement methodology for wireless sensor networks". In: *Proceedings of the international multiconference of engineers and computer scientists*. Vol. 1, pp. 18–20.

Bi, Zhuming, Li Da Xu, and Chengen Wang (2014). "Internet of things for enterprise systems of modern manufacturing". In: *IEEE Transactions on industrial informatics* 10.2, pp. 1537–1546.

Bittencourt, L F et al. (2015). "Towards Virtual Machine Migration in Fog Computing". In: *2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*, pp. 1–8. DOI: 10.1109/3PGCIC.2015.85.

Bittencourt, Luiz Fernando et al. (2015). "Towards virtual machine migration in fog computing". In: *P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2015 10th International Conference on*. IEEE, pp. 1–8.

Blackburn, M and G Grid (2008). "Five ways to reduce data center server power consumption". In: *The Green Grid*.

Bonomi, Flavio et al. (2012a). *Fog computing and its role in the internet of things*. Helsinki, Finland. DOI: 10.1145/2342509.2342513.

— (2012b). "Fog computing and its role in the internet of things". In: *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, pp. 13–16.

Borrego, Maura, Elliot P Douglas, and Catherine T Amelink (2009). "Quantitative, qualitative, and mixed research methods in engineering education". In: *Journal of Engineering education* 98.1, pp. 53–66.

Botella, Cristina et al. (2009). "An e-health system for the elderly (Butler Project): A pilot study on acceptance and satisfaction". In: *CyberPsychology & Behavior* 12.3, pp. 255–262.

Botta, Alessio et al. (2016). "Integration of cloud computing and internet of things: a survey". In: *Future Generation Computer Systems* 56, pp. 684–700.

Boyabatli, Onur and Ihsan Sabuncuoglu (2004). "Parameter selection in genetic algorithms". In: *Journal of Systemics, Cybernetics and Informatics* 4.2, p. 78.

Brewer, Eric A (2015). "Kubernetes and the path to cloud native". In: *Proceedings of the Sixth ACM Symposium on Cloud Computing*. ACM, pp. 167–167.

Brownlee, Jason et al. (2007). "A note on research methodology and benchmarking optimization algorithms". In: *Complex Intelligent Systems Laboratory (CIS), Centre for Information Technology Research (CITR), Faculty of Information and Communication Technologies (ICT), Swinburne University of Technology, Victoria, Australia, Technical Report ID* 70125.

Burkard, Rainer E et al. (1998). "The quadratic assignment problem". In: *Handbook of combinatorial optimization*. Springer, pp. 1713–1809.

Chaâri, Rihab et al. (2016). "Cyber-physical systems clouds: A survey". In: *Computer Networks* 108, pp. 260–278. DOI: http://dx.doi.org/10.1016/j.comnet.2016.08.017.

Chao, Kuo-Ming et al. (2015). "Cloud E-learning for Mechatronics: CLEM". In: *Future Generation Computer Systems* 48, pp. 46–59.

Chen, Xu et al. (2016). "Efficient multi-user computation offloading for mobile-edge cloud computing". In: *IEEE/ACM Transactions on Networking* 24.5, pp. 2795–2808.

Christophe, Benoit et al. (2011). "The web of things vision: Things as a service and interaction patterns". In: *Bell labs technical journal* 16.1, pp. 55–61.

Cicirelli, Franco et al. (2017). "Edge Computing and Social Internet of Things for large-scale smart environments development". In: *IEEE Internet of Things Journal* 4662.c, pp. 1–15. ISSN: 23274662. DOI: 10.1109/JIOT.2017.2775739.

Cisco Systems (2016). "Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are". In: *Www.Cisco.Com*, p. 6.

Cruz, Mauro A. A. da et al. (2018). "A Reference Model for Internet of Things Middleware". In: *IEEE Internet of Things Journal* 4662.c, pp. 1–1. ISSN: 2327-4662. DOI: 10.1109/JIOT.2018.2796561. URL: http://ieeexplore.ieee.org/document/8267034/.

Dastjerdi, Amir Vahid and Rajkumar Buyya (2016). "Fog computing: Helping the Internet of Things realize its potential". In: *Computer* 49.8, pp. 112–116.

Datta, Soumya Kanti, Christian Bonnet, and Navid Nikaein (2014). "An IoT gateway centric architecture to provide novel M2M services". In: *Internet of Things (WF-IoT), 2014 IEEE World Forum on*. IEEE, pp. 514–519.

Deng, R et al. (2016). "Optimal Workload Allocation in Fog-Cloud Computing Toward Balanced Delay and Power Consumption". In: *IEEE Internet of Things Journal* 3.6, pp. 1171–1181. DOI: 10.1109/JIOT.2016.2565516.

Dhinesh Babu, L D and P Venkata Krishna (2013). "Honey bee behavior inspired load balancing of tasks in cloud computing environments". In: *Applied Soft Computing* 13.5, pp. 2292–2303. DOI: http://dx.doi.org/10.1016/j.asoc.2013.01.025.

Díaz, Manuel, Cristian Martín, and Bartolomé Rubio (2016). "State-of-the-art, challenges, and open issues in the integration of Internet of things and cloud computing". In: *Journal of Network and Computer Applications* 67, pp. 99–117. ISSN: 10848045. DOI: 10.1016/j.jnca.2016.01.010.

DIN (2016). *German Standardization Roadmap – Industry 4.0 (Version 2)*.

Distefano, Salvatore, Giovanni Merlino, and Antonio Puliafito (2015). "A utility paradigm for IoT: The sensing Cloud". In: *Pervasive and mobile computing* 20, pp. 127–144.

Do, Cuong T. et al. (2015). "A proximal algorithm for joint resource allocation and minimizing carbon footprint in geo-distributed fog computing". In: *2015 International Conference on Information Networking (ICOIN)*. IEEE, pp. 324–329. ISBN: 978-1-4799-8342-1. DOI: 10.1109/ICOIN.2015.7057905.

Duro, João A, Robin C Purshouse, and Peter J Fleming (2018). "Collaborative Multi-Objective Optimization for Distributed Design of Complex Products". In:

Eclipse Kura (n.d.). https://www.eclipse.org/kura/. Accessed: 2019-12-22.

Ester, Martin et al. (1996). "A density-based algorithm for discovering clusters in large spatial databases with noise." In: *Kdd*. Vol. 96. 34, pp. 226–231.

Fortino, Giancarlo et al. (2014a). "Integration of agent-based and cloud computing for the smart objects-oriented IoT". In: *Proceedings of the 2014 IEEE 18th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*. IEEE, pp. 493–498.

— (2014b). "Middlewares for Smart Objects and Smart Environments: Overview and Comparison". In: *Internet of Things Based on Smart Objects: Technology, Middleware and Applications*. Ed. by Giancarlo Fortino and Paolo Trunfio. Cham: Springer International Publishing, pp. 1–27. ISBN: 978-3-319-00491-4.

Fox, Geoffrey C., Supun Kamburugamuve, and Ryan D. Hartman (2012). "Architecture and measured characteristics of a cloud based internet of things". In: *Proceedings of the 2012 International Conference on Collaboration Technologies and Systems, CTS 2012*, pp. 6–12. DOI: 10.1109/CTS.2012.6261020.

Al-Fuqaha, Ala et al. (2015). "Toward better horizontal integration among IoT services". In: *IEEE Communications Magazine* 53.9, pp. 72–79.

García-Valls, Marisol, Tommaso Cucinotta, and Chenyang Lu (2014). "Challenges in real-time virtualization and predictable cloud computing". In: *Journal of Systems Architecture* 60.9, pp. 726–740. DOI: http://dx.doi.org/10.1016/j.sysarc.2014.07.004.

Giurgiu, Ioana et al. (2009). "Calling the cloud: Enabling mobile phones as interfaces to cloud applications". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 5896 LNCS, pp. 83–102. ISSN: 03029743. DOI: 10.1007/978-3-642-10445-9_5.

Gubbi, Jayavardhana et al. (2013). "Internet of Things (IoT): A vision, architectural elements, and future directions". In: *Future generation computer systems* 29.7, pp. 1645–1660.

Gupta, Rushitaa and Raghav Garg (2015). "Mobile Applications Modelling and Security Handling in Cloud-Centric Internet of Things". In: *Proceedings - 2015 2nd IEEE International Conference on Advances in Computing and Communication Engineering, ICACCE 2015*, pp. 285–290. DOI: 10.1109/ICACCE.2015.119.

Gyrard, Amelie et al. (2015). "A Semantic Engine for Internet of Things: Cloud, Mobile Devices and Gateways". In: *Proceedings - 2015 9th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, IMIS 2015*, pp. 336–341. DOI: 10.1109/IMIS.2015.83.

Hakiri, Akram et al. (2015). "Publish/subscribe-enabled software defined networking for efficient and scalable IoT communications". In: *IEEE communications magazine* 53.9, pp. 48–54.

Hauke, Jan and Tomasz Kossowski (2011). "Comparison of values of Pearson's and Spearman's correlation coefficients on the same sets of data". In: *Quaestiones geographicae* 30.2, pp. 87–93.

He, X et al. (2016). "A novel load balancing strategy of software-defined cloud/fog networking in the Internet of Vehicles". In: *China Communications* 13.Supplement2, pp. 140–149. DOI: 10.1109/CC.2016.7833468.

Health and Safety Executive (2004). *Health and safety in engineering workshops*.

Heller, Brandon, Rob Sherwood, and Nick McKeown (2012). "The controller placement problem". In: *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, pp. 7–12.

Hemminger, Stephen (2005). "Network Emulation with NetEm". In: URL: https://www.rationali.st/blog/files/20151126-jittertrap/netem-shemminger.pdf.

Hong, Kirak et al. (2013). "Mobile fog: A programming model for large-scale applications on the internet of things". In: *Proceedings of the second ACM SIGCOMM workshop on Mobile cloud computing*. ACM, pp. 15–20.

Hoque, Saiful et al. (2017). "Towards Container Orchestration in Fog Computing Infrastructures". In: *Proceedings - International Computer Software and Applications Conference* 2, pp. 294–299. ISSN: 07303157. DOI: 10.1109/COMPSAC.2017.248.

Hossain, M Shamim et al. (2012). "Resource allocation for service composition in cloud-based video surveillance platform". In: *Multimedia and Expo Workshops (ICMEW), 2012 IEEE International Conference on*. IEEE, pp. 408–412.

Hu, J et al. (2010). "A Scheduling Strategy on Load Balancing of Virtual Machine Resources in Cloud Computing Environment". In: *2010 3rd International Symposium on Parallel Architectures, Algorithms and Programming*, pp. 89–96. DOI: 10.1109/PAAP.2010.65.

Intharawijitr, Krittin, Katsuyoshi Iida, and Hiroyuki Koga (2016). "Analysis of fog model considering computing and communication latency in 5G cellular networks". In: *2016*

IEEE International Conference on Pervasive Computing and Communication Workshops, PerCom Workshops 2016, pp. 5–8. DOI: 10.1109/PERCOMW.2016.7457059.

Inzinger, Christian et al. (2014). "MADCAT: A methodology for architecture and deployment of cloud application topologies". In: *Service Oriented System Engineering (SOSE), 2014 IEEE 8th International Symposium on*. IEEE, pp. 13–22.

Ismail, Bukhary Ikhwan et al. (2015). "Evaluation of docker as edge computing platform". In: *Open Systems (ICOS), 2015 IEEE Confernece on*. IEEE, pp. 130–135.

Iyer, Ravishankar K. and David J. Rossetti (1986). "A Measurement-Based Model for Workload Dependence of CPU Errors". In: *IEEE Transactions on Computers* C-35.6, pp. 511–519. ISSN: 0018-9340. DOI: 10.1109/TC.1986.5009428. URL: http://ieeexplore.ieee.org/document/5009428/.

Jalali, Fatemeh et al. (2016). "Fog computing may help to save energy in cloud computing". In: *IEEE Journal on Selected Areas in Communications* 34.5, pp. 1728–1739.

Jayaraman, Prem Prakash et al. (2014). "Cardap: A scalable energy-efficient context aware distributed mobile data analytics platform for the fog". In: *East European Conference on Advances in Databases and Information Systems*. Springer, pp. 192–206.

Jennings, Cullen, Jari Arkko, and Zach Shelby (2012). "Media types for sensor markup language (SENML)". In:

Jiang, Y (2016). "A Survey of Task Allocation and Load Balancing in Distributed Systems". In: *IEEE Transactions on Parallel and Distributed Systems* 27.2, pp. 585–599. DOI: 10.1109/TPDS.2015.2407900.

Jim Zw Li et al. (2011). "CloudOpt: multi-goal optimization of application deployments across a cloud". In: *Proceedings of the 7th International Conference on Network and Services Management*. International Federation for Information Processing, pp. 162–170. ISBN: 9783901882449. URL: https://dl.acm.org/citation.cfm?id=2147697.

Jingtao, Su et al. (2015). "Steiner tree based optimal resource caching scheme in fog computing". In: *China Communications* 12.8, pp. 161–168.

Khodadadi, Farzad, Rodrigo N Calheiros, and Rajkumar Buyya (2015). "A data-centric framework for development and deployment of Internet of Things applications in clouds". In: *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2015 IEEE Tenth International Conference on*. IEEE, pp. 1–6.

Kim, Donghyeon, Choonhwa Lee, and Sumi Helal (2015). "Enabling elastic services for OSGi-based cloud platforms". In: *Ubiquitous and Future Networks (ICUFN), 2015 Seventh International Conference on*. IEEE, pp. 407–409.

Kim, Seungryong, Chorwon Kim, and JongWon Kim (2017). "Reliable smart energy IoT-cloud service operation with container orchestration". In: *Network Operations and Management Symposium (APNOMS), 2017 19th Asia-Pacific*. IEEE, pp. 378–381.

Kim, Seungryong, Chorwon Kim, and Jongwon Kim (2017). "Operation with Container Orchestration". In: pp. 378–381.

Kimak, Stefan and Jeremy Ellman (2013). "Performance testing and comparison of client side databases versus server side". In: *Northumbria University*.

Kleinberg, Jon M et al. (1999). "The web as a graph: Measurements, models, and methods". In: *International Computing and Combinatorics Conference*. Springer, pp. 1–17.

Koschel, Arne et al. (2012). "Asynchronous messaging for OSGi". In: *Journal of computing and information technology* 20.3, pp. 151–157.

Kovatsch, Matthias, Yassin N Hassan, and Simon Mayer (2015). "Practical semantics for the Internet of Things: Physical states, device mashups, and open questions". In: *Internet of Things (IOT), 2015 5th International Conference on the*. IEEE, pp. 54–61.

Kovatsch, Matthias, Martin Lanter, and Simon Duquennoy (2012). "Actinium: A restful runtime container for scriptable internet of things applications". In: *Internet of Things (IOT), 2012 3rd International Conference on the*. IEEE, pp. 135–142.

Kum, Seung Woo et al. (2015). "A novel design of {IoT} cloud delegate framework to harmonize cloud-scale {IoT} services". In: *2015 {IEEE} {International} {Conference} on {Consumer} {Electronics} ({ICCE})*, pp. 247–248. DOI: 10.1109/ICCE.2015.7066399.

Kunz, T (1991). "The influence of different workload descriptions on a heuristic load balancing scheme". In: *IEEE Transactions on Software Engineering* 17.7, pp. 725–730. DOI: 10.1109/32.83908.

Lampesberger, Harald (2016). "Technologies for Web and cloud service interaction: a survey". In: *Service Oriented Computing and Applications* 10.2, pp. 71–110.

Lasi, Heiner et al. (2014). "Industry 4.0". In: *Business & Information Systems Engineering* 6.4, pp. 239–242.

Lawler, Eugene L (1963). "The quadratic assignment problem". In: *Management science* 9.4, pp. 586–599.

Lee, Gunho, Byung-Gon Chun, and H Katz (2011). *Heterogeneity-aware resource allocation and scheduling in the cloud*. Portland, OR.

Lee, Jay, Behrad Bagheri, and Hung-An Kao (2015). "A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems". In: *Manufacturing Letters* 3, pp. 18–23. DOI: http://dx.doi.org/10.1016/j.mfglet.2014.12.001.

Lee, Wangbong et al. (2016). "A gateway based fog computing architecture for wireless sensors and actuator networks". In: *Advanced Communication Technology (ICACT), 2016 18th International Conference on*. IEEE, pp. 210–213.

Li, Jim et al. (2009). "Performance model driven QoS guarantees and optimization in clouds". In: *2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*. IEEE, pp. 15–22. ISBN: 978-1-4244-3713-9. DOI: 10.1109/CLOUD.2009.5071528. URL: http://ieeexplore.ieee.org/document/5071528/.

Li, Zhe (2016). "COAST: A Connected Open plAtform for Smart objecTs". In: *Proceedings of the 2015 2nd International Conference on Information and Communication Technologies for Disaster Management, ICT-DM 2015*, pp. 166–172. DOI: 10.1109/ICT-DM.2015.7402060.

Lu, Yang (2017). "Industry 4.0: A survey on technologies, applications and open research issues". In: *Journal of Industrial Information Integration* 6, pp. 1–10. ISSN: 2452414X. DOI: 10.1016/j.jii.2017.04.005.

Lucas-Simarro, Jose Luis et al. (2013). "Scheduling strategies for optimal service deployment across multiple clouds". In: *Future Generation Computer Systems* 29.6, pp. 1431–1441. DOI: http://dx.doi.org/10.1016/j.future.2012.01.007.

Mahmud, Redowan and Rajkumar Buyya (2016). "Fog Computing: A Taxonomy, Survey and Future Directions". In: arXiv: 1611.05539.

Mell, Peter and Timothy Grance (2011). *The NIST Definition of Cloud Computing*. Tech. rep.

Merkel, Dirk (2014). "Docker: lightweight linux containers for consistent development and deployment". In: *Linux Journal* 2014.239, p. 2.

Minh, Quang Tran et al. (2017). "Toward service placement on fog computing landscape". In: *2017 4th NAFOSTED Conference on Information and Computer Science, NICS 2017 - Proceedings* 2017-Janua, pp. 291–296. DOI: 10.1109/NAFOSTED.2017.8108080.

Newman, Mark EJ (2003). "The structure and function of complex networks". In: *SIAM review* 45.2, pp. 167–256.

Nierbeck, Achim et al. (2014). *Apache Karaf Cookbook*. Packt Publishing Ltd.

Ningning, S et al. (2016). "Fog computing dynamic load balancing mechanism based on graph repartitioning". In: *China Communications* 13.3, pp. 156–164. DOI: 10.1109/CC.2016.7445510.

Nopiah, ZM et al. (2010). "Time complexity analysis of the genetic algorithm clustering method". In: *Proceedings of the 9th WSEAS International Conference on Signal Processing, Robotics and Automation, ISPRA*, pp. 171–176.

Orabi, Mahmoud Husseini, Ahmed Husseini Orabi, and Timothy Lethbridge (2016). "Umple as a component-based language for the development of real-time and embedded applications". In: *Model-Driven Engineering and Software Development (MODELSWARD), 2016 4th International Conference on*. IEEE, pp. 282–291.

Osanaiye, Opeyemi et al. (2017). "From cloud to fog computing: A review and a conceptual live VM migration framework". In: *IEEE Access* 5, pp. 8284–8300.

Oueis, Jessica, Emilio Calvanese Strinati, and Sergio Barbarossa (2015). "The fog balancing: Load distribution for small cell cloud computing". In: *Vehicular Technology Conference (VTC Spring), 2015 IEEE 81st*. IEEE, pp. 1–6.

Paraiso, Fawaz et al. (2012). "A federated multi-cloud PaaS infrastructure". In: *Proceedings - 2012 IEEE 5th International Conference on Cloud Computing, CLOUD 2012*, pp. 392–399. ISSN: 2159-6182. DOI: 10.1109/CLOUD.2012.79. arXiv: 1008.1900.

Pearson, Karl (1895). "Note on regression and inheritance in the case of two parents". In: *Proceedings of the Royal Society of London* 58, pp. 240–242.

Pereira, Pablo Puñal et al. (2013). "Enabling cloud-connectivity for mobile internet of things applications". In: *Proceedings - 2013 IEEE 7th International Symposium on Service-Oriented System Engineering, SOSE 2013*, pp. 518–526. DOI: 10.1109/SOSE.2013.33.

Rahmani, Amir-Mohammad et al. (2015). "Smart e-health gateway: Bringing intelligence to internet-of-things based ubiquitous healthcare systems". In: *Consumer Communications and Networking Conference (CCNC), 2015 12th Annual IEEE*. IEEE, pp. 826–834.

Ramezani, Fahimeh, Jie Lu, and Farookh Khadeer Hussain (2014). "Task-Based System Load Balancing in Cloud Computing Using Particle Swarm Optimization". In: *International Journal of Parallel Programming* 42.5, pp. 739–754. DOI: 10.1007/s10766-013-0275-4. URL: http://dx.doi.org/10.1007/s10766-013-0275-4.

Ruckebusch, Peter et al. (2016). "Gitar: Generic extension for internet-of-things architectures enabling dynamic updates of network and application modules". In: *Ad Hoc Networks* 36, pp. 127–151.

Rui, Jiang and Sun Danpeng (2015). "Architecture Design of the Internet of Things Based on Cloud Computing". In: *2015 Seventh International Conference on Measuring Technology and Mechatronics Automation*, pp. 206–209. DOI: 10.1109/ICMTMA.2015.57.

Sargent, Robert G (2007). "Verification and validation of simulation models". In: *Simulation Conference, 2007 Winter*. IEEE, pp. 124–137.

Sarkar, Chayan et al. (2015). "DIAT : A Scalable Distributed Architecture for IoT". In: 2.3, pp. 230–239.

Savazzi, Stefano, Vittorio Rampa, and Umberto Spagnolini (2014). "Wireless cloud networks for the factory of things: Connectivity modeling and layout design". In: *IEEE Internet of Things Journal* 1.2, pp. 180–195.

Scheuermann, Constantin, Stephan Verclas, and Bernd Bruegge (2015). "Agile factory-an example of an industry 4.0 manufacturing process". In: *Cyber-Physical Systems, Networks, and Applications (CPSNA), 2015 IEEE 3rd International Conference on*. IEEE, pp. 43–47.

Seo, Sangwon et al. (2015). "HePA: hexagonal platform architecture for smart home things". In: *Parallel and Distributed Systems (ICPADS), 2015 IEEE 21st International Conference on*. IEEE, pp. 181–189.

Singh, Meena et al. (2015). "Secure mqtt for internet of things (iot)". In: *Communication Systems and Network Technologies (CSNT), 2015 Fifth International Conference on*. IEEE, pp. 746–751.

Sivieri, Alessandro, Luca Mottola, and Gianpaolo Cugola (2016). "Building Internet of Things software with ELIoT". In: *Computer Communications* 89, pp. 141–153.

Skarlat, Olena, Bachmann Kevin, and Stefan Schulte (2018). "FogFrame: Service placement, deployment, and execution in the fog". In: *Future Generation Computer Systems*.

Spearman, Charles (1910). "Correlation calculated from faulty data". In: *British journal of psychology* 3.3, pp. 271–295.

Stojmenovic, Ivan (2014). "Fog computing: A cloud to the ground support for smart things and machine-to-machine networks". In: *Telecommunication Networks and Applications Conference (ATNAC), 2014 Australasian*. IEEE, pp. 117–122.

Taneja, Mohit and Alan Davy (2017). "Resource aware placement of IoT application modules in Fog-Cloud Computing Paradigm". In: *Proceedings of the IM 2017 - 2017 IFIP/IEEE International Symposium on Integrated Network and Service Management*, pp. 1222–1228. ISSN: 9783901882890. DOI: 10.23919/INM.2017.7987464.

Tao, Fei et al. (2014). "IoT-Based intelligent perception and access of manufacturing resource toward cloud manufacturing". In: *IEEE Transactions on Industrial Informatics* 10.2, pp. 1547–1557. DOI: 10.1109/TII.2014.2306397.

Trappey, A J C et al. (2016). "A Review of Technology Standards and Patent Portfolios for Enabling Cyber-Physical Systems in Advanced Manufacturing". In: *IEEE Access* 4, pp. 7356–7382. DOI: 10.1109/ACCESS.2016.2619360.

Truong, Hong-Linh and Schahram Dustdar (2015). "Principles for engineering IoT cloud systems". In: *IEEE Cloud Computing* 2.2, pp. 68–76.

Verba, Nandor, Kuo-Ming Chao, Anne James, Daniel Goldsmith, et al. (n.d.). "Platform as a service gateway for the Fog of Things". In: *Advanced Engineering Informatics*. DOI: http://dx.doi.org/10.1016/j.aei.2016.11.003.

Verba, Nandor, Kuo-Ming Chao, Anne James, Jacek Lewandowski, et al. (2017). "Graph Analysis of Fog Computing Systems for Industry 4.0". In: *e-Business Engineering (ICEBE), 2017 IEEE 14th International Conference on*. IEEE, pp. 46–53.

Verma, Prabal and Sandeep K Sood (2018). "Fog Assisted-IoT Enabled Patient Health Monitoring in Smart Homes". In: *IEEE Internet of Things Journal*.

Verma, S et al. (2016). "An efficient data replication and load balancing technique for fog computing environment". In: *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, pp. 2888–2895.

Vogler, Michael, Johannes M Schleicher, et al. (2015). "DIANE-dynamic IoT application deployment". In: *Mobile Services (MS), 2015 IEEE International Conference on*. IEEE, pp. 298–305.

Vogler, Michael, Johannes Schleicher, et al. (2016). "Optimizing elastic IoT application deployments". In: *IEEE Transactions on Services Computing*.

Vögler, Michael et al. (2016). "A scalable framework for provisioning large-scale IoT deployments". In: *ACM Transactions on Internet Technology (TOIT)* 16.2, p. 11.

Voutyras, Orfefs et al. (2015). "Social monitoring and social analysis in internet of things virtual networks". In: *Intelligence in Next Generation Networks (ICIN), 2015 18th International Conference on*. IEEE, pp. 244–251.

Wang, Congjie et al. (2017). "Optimizing Multi-Cloud CDN Deployment and Scheduling Strategies Using Big Data Analysis". In: *2017 IEEE International Conference on Services Computing (SCC)*, pp. 273–280. DOI: 10.1109/SCC.2017.42. URL: http://ieeexplore.ieee.org/document/8034995/.

Wang, Lihui, Martin Törngren, and Mauro Onori (2015). "Current status and advancement of cyber-physical systems in manufacturing". In: *Journal of Manufacturing Systems* 37, Part 2, pp. 517–527. DOI: http://dx.doi.org/10.1016/j.jmsy.2015.04.008.

Wang, Nan et al. (2017). "ENORM: A Framework For Edge NOde Resource Management". In: *IEEE Transactions on Services Computing* X.JANUARY, pp. 1–14. ISSN: 19391374. DOI: 10.1109/TSC.2017.2753775. arXiv: 1709.04061.

Wiesner, Stefan, Eugenia Marilungo, and Klaus-Dieter Thoben (2017). "Cyber-Physical Product-Service Systems: Challenges for Requirements Engineering (Mini Special Issue on Smart Manufacturing)". In: *International journal of automation technology* 11.1, pp. 17–28.

Wolpert, David H and William G Macready (1997). "No free lunch theorems for optimization". In: *IEEE transactions on evolutionary computation* 1.1, pp. 67–82.

Wu, Xiaonian et al. (2013). "A Task Scheduling Algorithm based on QoS-Driven in Cloud Computing". In: *Procedia Computer Science* 17, pp. 1162–1169. DOI: http://dx.doi.org/10.1016/j.procs.2013.05.148.

Xu, Rui and Donald Wunsch (2005). "Survey of clustering algorithms". In: *IEEE Transactions on neural networks* 16.3, pp. 645–678.

Zeng, Deze, Lin Gu, Song Guo, et al. (2016). "Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system". In: *IEEE Transactions on Computers* 65.12, pp. 3702–3712.

Zeng, Deze, Lin Gu, and Hong Yao (2018). "Towards energy efficient service composition in green energy powered Cyber-Physical Fog Systems". In: *Future Generation Computer Systems*, pp. 1–9. ISSN: 0167739X. DOI: 10.1016/j.future.2018.01.060. URL: https://doi.org/10.1016/j.future.2018.01.060.

Zhan, Zhi-Hui et al. (2015). "Cloud Computing Resource Scheduling and a Survey of Its Evolutionary Approaches". In: *ACM Comput. Surv.* 47.4, pp. 1–33. DOI: 10.1145/2788397.

Zhang, Yin et al. (2017). "Health-CPS: Healthcare cyber-physical system assisted by cloud and big data". In: *IEEE Systems Journal* 11.1, pp. 88–95.

Zhao, Jia et al. (2013). "A Location Selection Policy of Live Virtual Machine Migration for Power Saving and Load Balancing". In: *The Scientific World Journal* 2013, p. 16. DOI: 10.1155/2013/492615.

Appendix A

VisJs Visualisation Platform

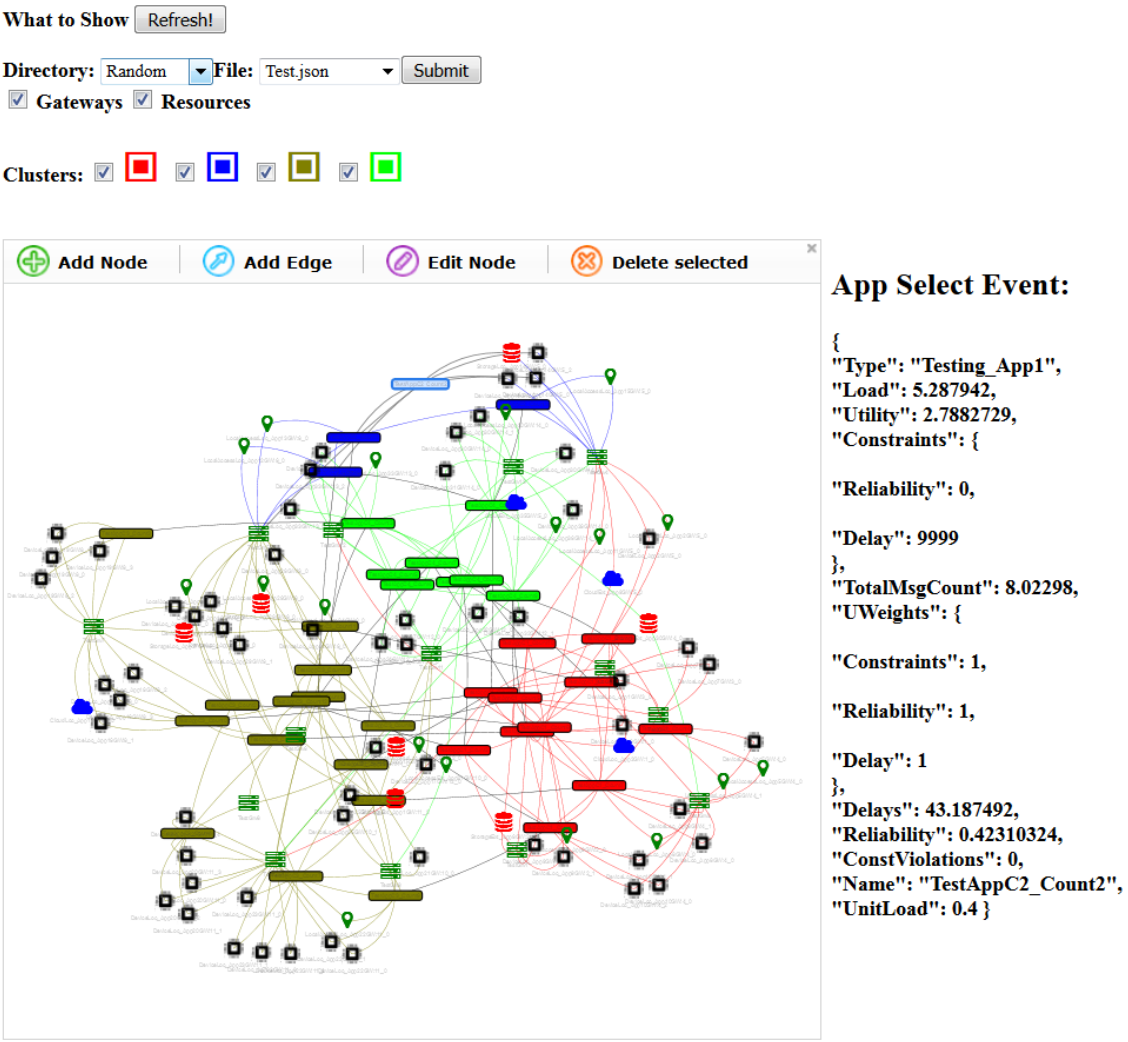

Fig. A.1 Vis.Js Platform

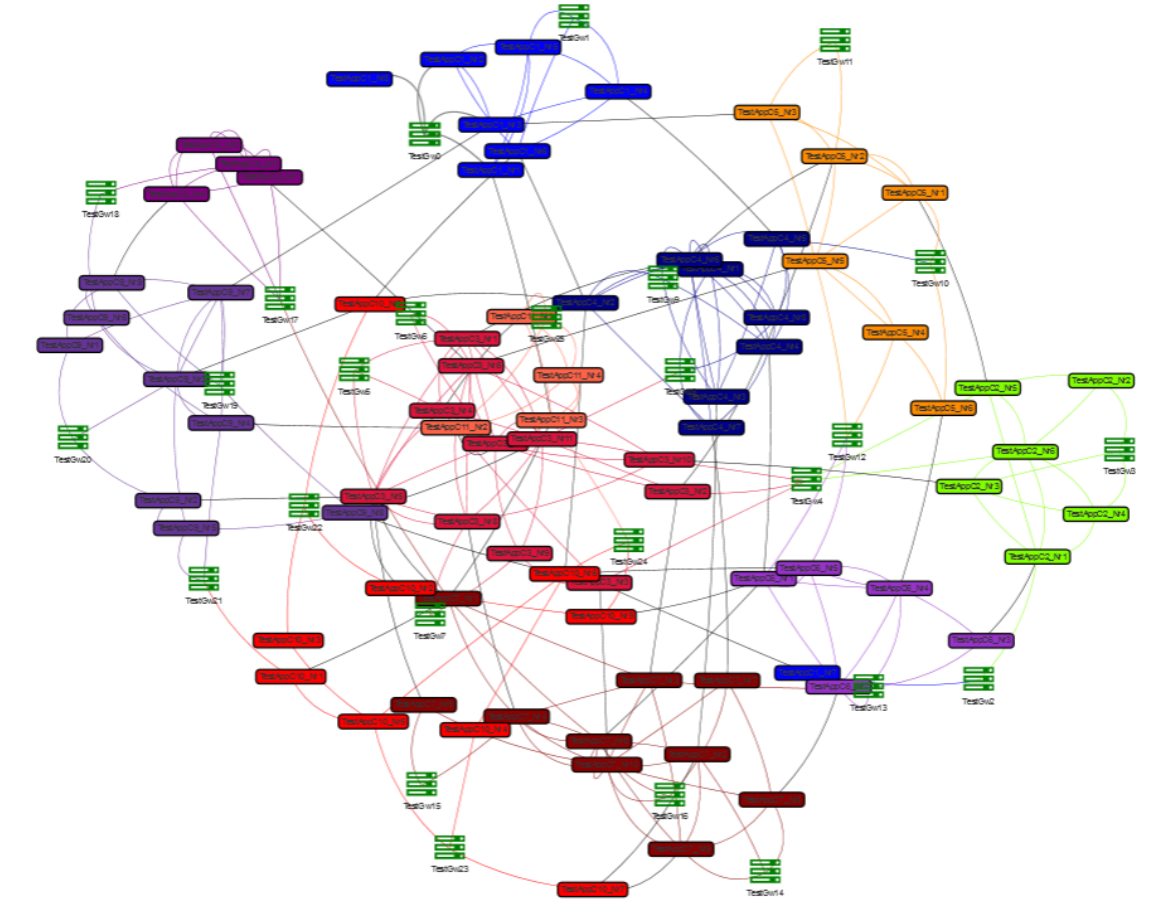

Fig. A.2 Initial Generated Fog

Vis.js dynamic browser library was modified to be able to view Fog Deployments together with application, gateway and resource parameters. This was used to see how certain methods perform and to have an overview of the system.

The modified platform can be seen in Fig. A.1 where the user can select directories and files from the hdfs clous storage system which can be used by the Spark cluster to save outputs as well. The users can select clusters to view and also to remove Gateways and Resources for faster loading. By clicking on entities they can view information on them.

The downside of the platform is that above 100 application loading the system is slow and details of it are hard to view.

The Fig. A.2 shows the generated Multi-Components fog scenarios for an application size of 80. The results is Fig. A.3 shows the results of the distance based clustering and deployment while the results in Fig. A.4 shows the results of the Sampling and Weights based methods results. It is worth noting that as the Vis.Js platform shows connection between components the clustering made by the first method seems to make sense on the platform

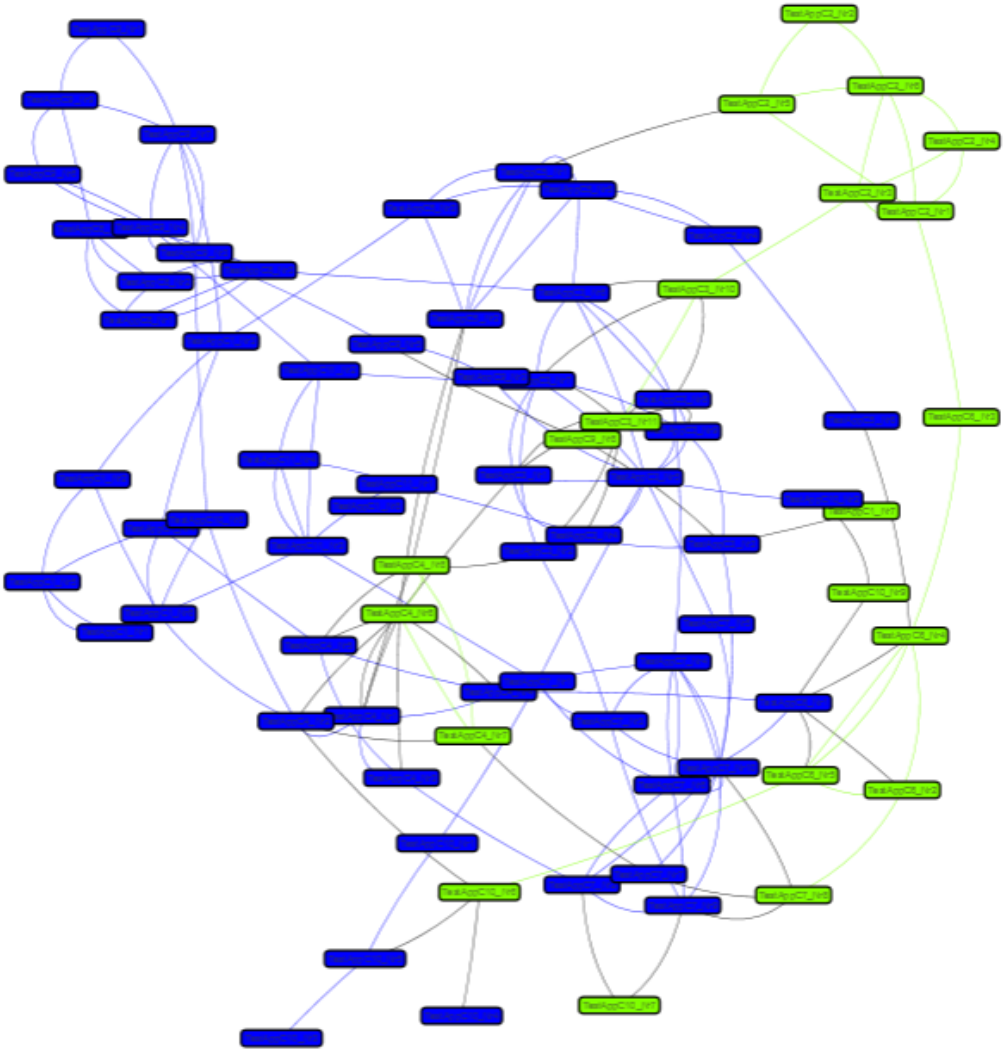

Fig. A.3 Results of Distance Deployment

while the one done by the second one seems random, but as seen from the tests has better results.

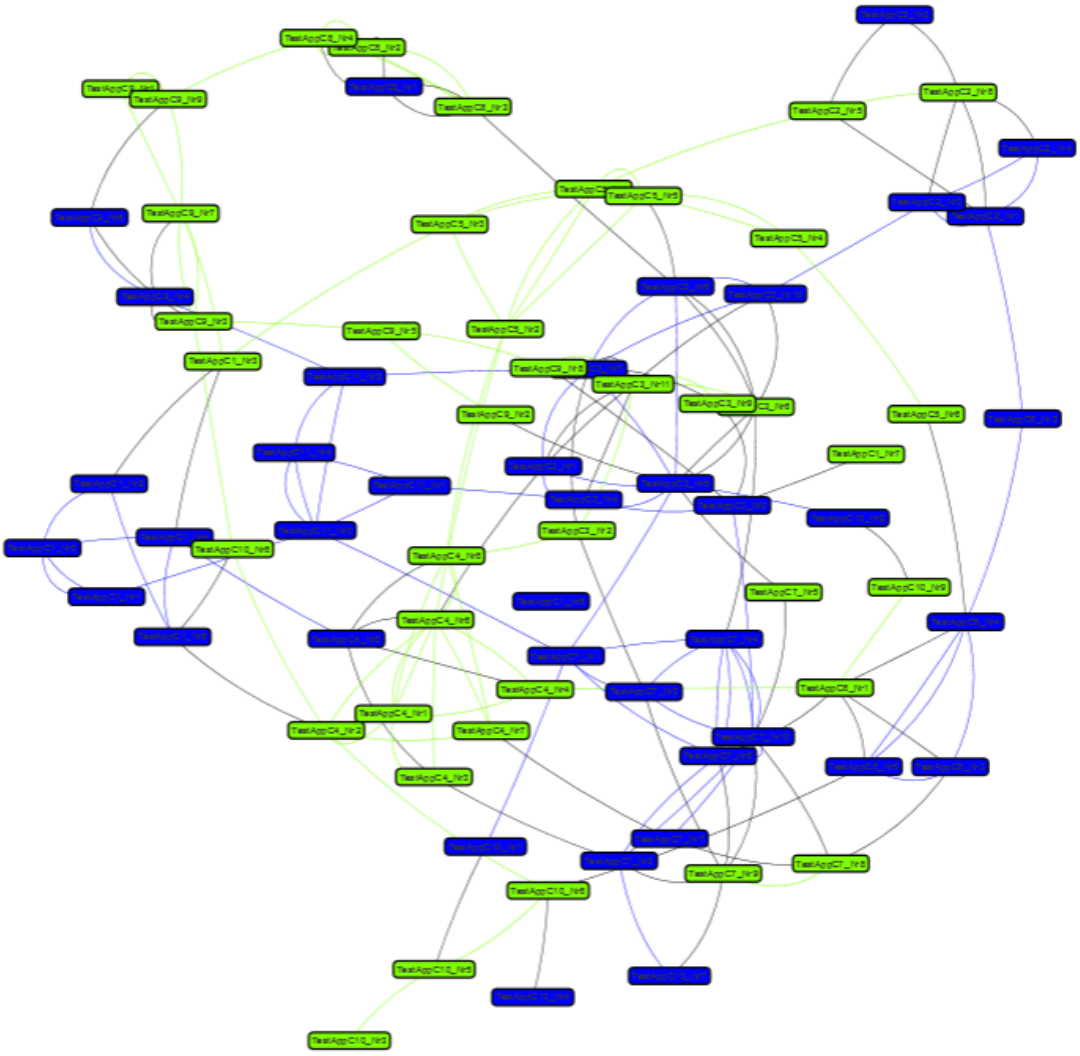

Fig. A.4 Results of Sampling and Weights Clustering Deployment

Appendix B

Code Snippets

This Appendix aims to show some of the main code snippets from the platform with components from the drivers, load and testing applications as well as parts from the monitoring drivers and from the java testing platform that was developed. The origin of the code is reflected in the title of the included sections.

The whole code for the drivers, the deployments and main components can be found at the GitHub Directory github.com/nandor1992/FogOfThings, where the Gateway components and Drivers can be found under the *master* branch and the optimization tests can be found under the *JavaUpdates* branch. This sections aims to highlight some of the more important components.

A sample of the Sampling and Weighted Clustering method can be seen in Fig. B.1. A portion of the testing application can be seen in Fig. B.2. The main section of the AMQP to Karaf Event Admin broker are presented in Fig. B.3 and a part of the BLuetooth Driver can be seen in Fig. B.4.

```
public static Map<Integer, Integer> SampleWeDiCOptimization(Fog f) {
        f.clearAppToGws();
        //Sampling
        float sampleProc = (float)0.2;
        int minSampleSize = getMinPts(f)*3;
        //Clustering
        int minPts = getMinPts(f);
        //Res Share
        int maxShare = 2;
        double shareThreshold = (float) 0.3;
        //GA
        int size = getMinPtsSize(minPts,f.getScenario());
        int count = getMinPtsCount(minPts,f.getScenario());
        //Time start
        long startIni = System.currentTimeMillis();
        dataG.addTime((float)(System.currentTimeMillis()-startIni)/(float)1000.0);
        dataG.addUtility((float)0.0);
        WeightedCls cls = new WeightedCls(f);
        Map<Integer,Integer> bestSolution = new HashMap<>();
        Double bestUtil = 0.0;
        cls.initTrain(10,2);
        //Random Population Initialization using Initial Weights
        List<Map<Integer, Integer>> bests =
        iterSampleClustGA(f,cls,getMinPtsCount((int)sampleProc*f.getApps().size(),f.getScenario()),getMinPtsSize((int)samplePro
        c*f.getApps().size(),f.getScenario()),sampleProc,minSampleSize);
        dataG.addTime((float)(System.currentTimeMillis()-startIni)/(float)1000.0);
        dataG.addUtility((float)0.0);
        if (bests == null){ System.out.println("Initial Random Clustering Failed!");return null;}
        bestUtil = getPartialUtility(f,bests.get(0)).doubleValue();
        bestSolution = bests.get(0);
        System.out.println("Sampling Best Util:"+bestUtil+" with solution: "+bestSolution);
        cls.getWeight().attemptResult(bestUtil.floatValue());
        System.out.println("Sampling Finished in :"+((System.currentTimeMillis()-startIni)/1000.0));

        dataG.addTime((float)(System.currentTimeMillis()-startIni)/(float)1000.0);
        dataG.addUtility((float)0.0);
        //Iterative Solution
        Map<Integer, Integer> best = IterWeDiCompOptimization(f, cls, bests, minPts, maxShare, shareThreshold, bestUtil,
        bestSolution,startIni);
        //Final Write Out
        return best
    }

    //iterSampleClustGA
    public List<Map<Integer, Integer>> sampleFogAttempt(){
        //Initial Attempt
        List<Map<Integer, Integer>> ret = attemptRandInstance();
        while (ret == null && failCnt<maxFailCnt){
            //First Attempt failed do diiiiive util size < min then do min while solution is found and if none is found escape
            System.out.println();
            System.out.println("----- Failed Attempt with clsSize: "+clsSize+" and Weights:" +appWeights+" - "+gwWeights+"
            FailCnt: "+failCnt);
            if (clsSize==minSize){
                failCnt++;

                this.modifyWeights(1.0-(1.0/(double)(2*maxFailCnt+1)*(double)failCnt),1.0+(1.0/(double)(2*maxFailCnt+1)*(double
                )failCnt));
                }
            if (clsSize/2<=minSize){clsSize=minSize;}else{clsSize=clsSize/2;}
            ret = attemptRandInstance();
        }
        List<Map<Integer, Integer>> tmpRet = ret;
        int refClsSize = clsSize;
        while (clsSize!=(int)(f.getApps().size()*proc) && failCnt<maxFailCnt){
            //Grow Until Clusterin can be done
            if (tmpRet!=null){
                ret=tmpRet;
                refClsSize = clsSize;
                System.out.println("----- Successfull Attempt with clsSize: "+clsSize);
                this.interpretWeights(ret);
                if (failCnt>1){

                    this.modifyWeights(1.0-(1.0/(double)(2*maxFailCnt+1)*(double)failCnt),1.0+(1.0/(double)(2*maxFailCnt+1)*(do
                    uble)failCnt));}
                double multi = 2.0;
                if (failCnt!=0){
                    multi = 2.0-failCnt/maxFailCnt;}
                if (clsSize*multi>(int)(f.getApps().size()*proc)){clsSize=(int)(f.getApps().size()*proc);}
```

```
                else{clsSize=(int)(clsSize*multi);}
                System.out.println("Cls Size: "+clsSize+" Max Fail: "+maxFailCnt+" FailCnt: "+failCnt+" and Weights:"
                +appWeights+" - "+gwWeights);
                tmpRet = attemptInstance();
            }else{
                System.out.println();
                System.out.println("----- Failed Attempt with clsSize: "+clsSize+" FailCnt: "+failCnt);
                failCnt++;
                clsSize=(int) (clsSize*(1.0-1/(double)maxFailCnt));
                if (clsSize<=refClsSize){break;}
                System.out.println("Cls Size: "+clsSize+" Max Fail: "+maxFailCnt+" FailCnt: "+failCnt+" and Weights:"
                +appWeights+" - "+gwWeights);

                this.modifyWeights(1.0-(1.0/(double)(2*maxFailCnt+1)*(double)failCnt),1.0+(1.0/(double)(2*maxFailCnt+1)*(double
                )failCnt));
                tmpRet = attemptInstance();
            }
        }
        return ret;
    }

    public static List<Map<Integer, Integer>> IterWeDiCompOptimization(Fog f, WeightedCls cls,
            List<Map<Integer, Integer>> bests, int minPts, int maxShare, double shareThreshold,
            Double bestUtil, Map<Integer, Integer> bestSolution,long startIni) {
        Map<String, Float> prog = new HashMap<String, Float>();
        boolean nextStep = true;
        //cls.getWeight().showData();
        // Loop here while Weighting Algorithm knows what to do next
        while (cls.getWeight().getNextStep()) {
            long start = System.currentTimeMillis();
            dataG.addUtility(bestUtil.floatValue());
            dataG.addTime((float)(System.currentTimeMillis()-startIni)/(float)1000.0);
            // Put values to the new weights Calculation
            weightsCorrBasedTraining(cls, bests);
            dataG.addWeightApp(cls.getWeight().appWeights());
            dataG.addWeightGw(cls.getWeight().gwWeights());
            System.out.println("Clustering Parameters: " + cls.getWeight().getChar() + " ----------");
            cls.getWeight().showWeights();
            // Try Clustering based on given weights If all eps failes then weights fail
            if (WeightedClustering(f, cls, minPts)) {
                dataG.addUtility(bestUtil.floatValue());
                dataG.addTime((float)(System.currentTimeMillis()-startIni)/(float)1000.0);
                weightedResourceAlloc(f, cls, maxShare, shareThreshold);
                dataG.addUtility(bestUtil.floatValue());
                dataG.addTime((float)(System.currentTimeMillis()-startIni)/(float)1000.0);
                // Do weighted Resource Allocation based algorithm; Do Local GA, if Any fail then the method fails
                List<Map<Integer, Integer>> tmpbests = Methods.GAClus(f, true,dataG);
                if (tmpbests == null) {
                    System.out.println("Direction Clustering Failed!");
                    cls.getWeight().setGwFailed();
                    dataG.addUtility(bestUtil.floatValue());
                    dataG.addTime((float)(System.currentTimeMillis()-startIni)/(float)1000.0);
                } else {
                    System.out.println("Direction Clustering Done in :" + ((System.currentTimeMillis() - start) / 1000.0));
                    bests = tmpbests;
                    prog.put("Clust[" + cls.getWeight().getChar() + " Time: "
                                    + ((System.currentTimeMillis() - start) / 1000.0) + "]",
                            getUtility(f, bests.get(0)));
                    if (getUtility(f, bests.get(0)) > bestUtil) {
                        bestUtil = getUtility(f, bests.get(0)).doubleValue();
                        bestSolution = bests.get(0);
                        dataG.setBestWeight(dataG.getCurrent());
                        dataG.setBestCluster(f.retreiveCluster());
                    }
                    dataG.addUtility(bestUtil.floatValue());
                    dataG.addTime((float)(System.currentTimeMillis()-startIni)/(float)1000.0);
                    cls.getWeight().attemptResult(getUtility(f, bests.get(0)));
                }
            } else {
                System.out.println("Direction Clustering Failed!");
                cls.getWeight().setAppFailed();
            }
        }
        System.out.println("Results: ");
        SortedSet<String> intKeys = new TreeSet<>(prog.keySet());
        for (String name : intKeys) {System.out.println(name + " = " + prog.get(name));}
        return bests
    }
```


Fig. B.1 Sampling and Weights Algorithm Snippet

```
    private static void sendEvent(String proc_time,String start) {
        Thread.currentThread().setName(name);
        Dictionary props = new Hashtable();
        props.put("app", name);
        props.put("payload","{'proc_time':"+proc_time+",'start_time':"+start+"}");
        props.put("device",device);
        Event event = new Event(regs.get(0), props);
        ea.sendEvent(event);
    }

    @Override
    public void updated(Dictionary properties) throws ConfigurationException {
        logger.warn("App Updated");
        device = properties.get("device").toString().trim();
        load = Integer.parseInt(properties.get("load").toString().trim());
        if (sr2!=null){
            sr2.unregister();
        }
        Dictionary dic = new Hashtable();
        dic.put(EventConstants.EVENT_TOPIC, regs.get(1)+device);
        sr2 = bcontext.registerService(EventHandler.class.getName(), this, dic);
    }

    @Override
    public void handleEvent(Event event) {
        Thread.currentThread().setName(name);
        String start = event.getProperty("payload").toString();
        //Do Load Then Respond
        long timeBefore = System.nanoTime();
        l1.doMatrice(load);
        l1.doFiltering(load);
        l1.doFlops(load);
        long timeAfter = System.nanoTime();
        long elapsed_time = timeAfter - timeBefore;
        //Respond
        sendEvent(String.valueOf(elapsed_time/1000), start);
    }

    public class Load {
    public void doMatrice(int qty){
        for( int i=0;i<qty;i++){
            int m1[][];
            int m2[][];
            int tmp[][];
            m1=randMatrice();
            m2=randMatrice();
            tmp=addMatrice(m1, m2);
            tmp=multMatrice(m1, m2);
        }
    }
    public int[][] randMatrice(){
        int m1[][];
        Random rand = new Random();
        m1=new int[20][20];
        for (int i=0;i<20;i++){
            for (int j=0;j<20;j++){
                m1[i][j] = rand.nextInt(200);
            }
        }
        return m1;
    }
    public int[][] addMatrice(int[][] a, int[][] b)
    {
        int ret[][];
        ret=new int[20][20];
        for (int i=0;i<20;i++){
            for (int j=0;j<20;j++){
                ret[i][j]=a[i][j]+b[i][j];
            }
        }
        return ret;
    }
    public int[][] multMatrice(int[][] a, int[][] b)
    {
        int ret[][];
        ret=new int[20][20];
        for (int i=0;i<20;i++){
            for (int j=0;j<20;j++){
```

```
                for (int k=0;k<20;k++)
                {
                    ret[i][j]=ret[i][j]+a[i][k]+b[k][j];
                }
            }
        }
        return ret;
    }
    public void displayMatrice(int[][] a)
    {
        System.out.println("Displaying Matrice");
        for (int i=0;i<20;i++){
            for (int j=0;j<20;j++){
                System.out.print(a[i][j]+" ");
            }
            System.out.println();
        }
    }
    /*------------Filtering Operations-----------*/
    public void doFiltering(int qty){
        for( int i=0;i<qty;i++){
            int data[];
            int minmax[]=new int[2];
            data=randData();
            float sma[]=smaData(data);
            float avg=avgData(data);
            minmax=minmaxData(data);
            sortData(data,true);
            sortData(data,false);
        }
    }
    public int[] sortData(int data[],boolean type){
        int tmp=0;
        for (int i=0;i<99;i++)
        {
            for (int j=i;j<100;j++)
            {
                if (!(data[i]>data[j] ^ type)){
                    tmp=data[j];
                    data[j]=data[i];
                    data[i]=tmp;
                }
            }
        }
        return data;
    }
    public float avgData(int data[]){
        int avg=0;
        for (int i=0;i<100;i++){
            avg=avg+data[i]/100;
        }
        return avg;
    }
    public int[] minmaxData(int data[]){
        int ext[]=new int[2];
        ext[0]=data[0];
        ext[1]=data[0];
        for (int i=1;i<100;i++){
            if (data[i]<ext[0]){
                ext[0]=data[i];
            }
            if (data[i]>ext[1]){
                ext[1]=data[i];
            }
        }
        return ext;
    }
    public float[] smaData(int data[]){
        ///Simple moving average
        float sma[] = new float[100];
        float sma_tmp=0;
        int curr=4;
        //Length of sma =5;
        for (int i=0;i<5;i++)
        {
            sma_tmp=sma_tmp+data[i];
        }
        sma_tmp=sma_tmp/5;
```

```
        sma[4]=sma_tmp;
        sma[0]=sma_tmp;
        sma[1]=sma_tmp;
        sma[2]=sma_tmp;
        sma[3]=sma_tmp;
        while (curr<99){
            sma_tmp=sma_tmp-(data[curr-4]/5)+(data[curr+1]/5);
            sma[curr]=sma_tmp;
            curr++;
        }
        sma[99]=sma_tmp;
        return sma;
    }
    public int[] randData(){
        int data[];
        Random rand = new Random();
        data=new int[100];
        for (int i=0;i<100;i++)
        {
            data[i]=rand.nextInt(1024);
        }
        return data;
    }
    public void printData(int data[]){
        System.out.println("Printing Data");
        for (int i=0;i<100;i++){
            System.out.print(data[i]+" ");
        }
        System.out.println();
    }
    public void printData(float data[]){
        System.out.println("Printing Data");
        int len=data.length;
        for (int i=0;i<len;i++){
            System.out.print(String.format("%5.4f",data[i])+" ");
        }
        System.out.println();
    }
    /*--------------Part For Flops--------------*/
    public void doFlops(int qty){
        for( int i=0;i<qty;i++){
            float d1[];
            float d2[];
            d1=randFloat();
            d2=randFloat();
            float ret[][]=doFloatOps(d1, d2);
        }
    }
    public float[][] doFloatOps(float[] d1,float[] d2)
    {
        float ret[][]=new float[4][1000];
        for (int i=0; i<1000;i++){
            ret[0][i]=d1[i]+d2[i];
            ret[1][i]=d1[i]-d2[i];
            ret[2][i]=d1[i]*d2[i];
            ret[3][i]=d1[i]/d2[i];
        }
        return ret;
    }
    public float[] randFloat(){
        float data[];
        Random rand = new Random();
        data=new float[1000];
        for (int i=0;i<1000;i++)
        {
            data[i]=rand.nextFloat();
        }
        return data;
    }
}
```


Fig. B.2 Testing App and Load generator Snippet

```
        ConnectionFactory factory = new ConnectionFactory();
        factory.setUsername("admin");
        factory.setVirtualHost("test");
        factory.setHost("localhost");
        factory.setPort(5672);
        try {
            connection = factory.newConnection();
            channel = connection.createChannel();
            channel.basicQos(1);
        } catch (IOException e1) {
            e1.printStackTrace();
        }
        Consumer consumer = new DefaultConsumer(channel) {
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties,
                    byte[] body) throws IOException {
                String message = new String(body, "UTF-8");
                Map<String, Object> headers = properties.getHeaders();
                // display time and date using toString()
                try {
                    sendEvent(message, headers);
                } catch (InterruptedException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
                channel.basicAck(envelope.getDeliveryTag(), false);
            }
        };
        try {
            channel.basicConsume(QUEUE_NAME, false, consumer);
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
    private void sendEvent(String message, Map<String, Object> headers) throws InterruptedException {
        Dictionary props = new Hashtable();
        for (Map.Entry<String, Object> header : headers.entrySet()) {
            props.put(header.getKey(), header.getValue());
        }
        props.put("payload", message);
        if (props.get("device")!=null) {
            Event event = new Event(DEVICE_QUEUE + props.get("device"), props);
            ea.sendEvent(event);
        }else if (props.get("cloud")!=null) {
            Event event = new Event(CLOUD_QUEUE + props.get("app"), props);
            ea.sendEvent(event);
        }else if (props.get("region")!=null) {
            Event event = new Event(REGION_QUEUE + props.get("app"), props);
            ea.sendEvent(event);
        }else if (props.get("res")!=null) {
            Event event = new Event(RESOURCE_QUEUE + props.get("app"), props);
            ea.sendEvent(event);
        }else if (props.get("app")!=null){
            if (props.get("app_type")!=null && props.get("app_rec")!=null){
                if (props.get("app_type").toString().equals("receive")){
                    String app=props.get("app_rec").toString();
                    props.remove("app_type");
                    props.remove("app_rec");
                    Event event = new Event(APP_REC_QUEUE + app, props);
                    ea.sendEvent(event);
                }else if (props.get("app_type").toString().equals("send")) {
                    String app=props.get("app_rec").toString();
                    props.remove("app_type");
                    props.remove("app_rec");
                    Event event = new Event(APP_SEND_QUEUE + app, props);
                    ea.sendEvent(event);
                }
            }
        }
    }
```


Fig. B.3 AMQP to Event Admin Broker Snippet

```
class blue(Daemon):
    class BlueThread(threading.Thread):
        def transLoop(self,socket,send_q,receive_q):
            message="";
            while not exitFlag:
                if not send_q.empty():
                    data = send_q.get()
                    logging.debug("Sending: " +data)
                    socket.send(data+'\n')
                try:
                    reading=socket.recv(1024)
                    if ((len(reading)>0 and reading[0]!=' ')):
                        message=message+reading;
                    if message[-1]=='\n' and message[-2]=='}':
                        receive_q.put(message)
                        message="";
                    if (len(message)>250):
                        logging.debug("Long Message Error")
                        logging.debug(len(message))
                        message=""
                except bluetooth.BluetoothError as error:
                    if error[0]!="timed out":
                        logging.error("Error:",error)
                        break;
            socket.close()

    class ServerThread(threading.Thread):
        def __init__(self,server_sock):
            threading.Thread.__init__(self)
            self.server_sock=server_sock
        def run(self):
            logging.debug("Started Server Thread")
            port = self.server_sock.getsockname()[1]
            logging.debug("Waiting for connection on RFCOMM channel %d" % port)
            self.server_sock.settimeout(2)
            while not exitFlag:
                try:
                    client_socket, client_info = self.server_sock.accept()
                    client_socket.send('x')
                    addr=client_info[0]
                    client_socket.settimeout(0.1)
                    send_q[addr]=Queue.Queue(10)
                    receive_q[addr]=Queue.Queue(10)
                    bluet=blue.BlueThread(addr,client_socket,send_q[addr],receive_q[addr])
                    bluet.start()
                    threads[addr]=bluet
                except bluetooth.BluetoothError :
                    pass
            logging.debug("Exiting Server")
    def messageResolv(self,my_json):
        dev_id=my_json.get("bn")[11:]
        dev_id=dev_id[:8]
        logging.debug(dev_id)
        if (self.dev_list.count(dev_id)!=0):
            message_amqp=json.dumps(my_json["e"])
            message_amqp=message_amqp.replace('"',"'")
            logging.debug(message_amqp)
            if dev_id in self.timer_list:
                self.timer_list[dev_id].cancel()
            properties_m=pika.BasicProperties(headers={'device':""+dev_id,'comm':
            ""+self.gw_name,'datetime':""+datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")})
            self.channel.basic_publish(exchange='device', routing_key='', body=message_amqp, properties=properties_m)
            self.updateDevTime(dev_id,"Available")
        else:
            logging.debug("No details found - not forwarding")

    def registerResolv(self,key,my_json):
        trans=my_json.get("bn",{})[2]
        my_uuid=self.resolv_dev(my_json)
        trans=trans[9:]
        data="{'bn':['trans:id:"+trans+"','urn:dev:id:"+my_uuid+"']}"
        self.dev_mac[my_uuid]=key
        self.sendRf(my_uuid,data);

    def resolv_dev(self,my_json):
        logging.debug("Json Parts!")
        type_d=my_json.get("bn",{})[0][13:]
        mac_d=my_json.get("bn",{})[1][12:]
```

```
        ver_d=my_json.get("ver")
        value=self.datab.lookupDev(mac_d,type_d,ver_d)
        if value!=None:
            logging.debug("Found details "+value)
            rand_uuid=value
            self.updateDevTime(value,"Available")
            self.dev_list.append(value)
        else:
            logging.debug("No details found")
            rand_uuid = ''.join([random.choice(string.ascii_letters+string.digits) for n in xrange(8)])
            sense=[]
            for sens in my_json["e"]:
                sense.append({sens["n"]:sens["u"]})
            self.datab.addDevice(rand_uuid,type_d,mac_d,ver_d,datetime.datetime.now().strftime("%Y-%m-%d
            %H:%M:%S"),'Available',sense)
            self.dev_list.append(rand_uuid)
        return rand_uuid

    def sendMssgResolv(self,data,header):
        logging.debug("---------Send Data---------")
        logging.debug(data)
        dev_id=header.headers.get('device')
        qos=str(header.headers.get('qos')).strip()
        logging.debug(dev_id)
        if (self.dev_list.count(dev_id)!=0):
            logging.debug("Found details")
            send_json="{'e':"+data+",'bn':'urn:dev:id:"+dev_id+"'}"
            send_json=send_json.replace(" ","_")
            logging.debug(send_json)
            #Start timer here
            if (qos!=None):
                if (qos=='1'):
                    t=Timer(5,self.timeout,[dev_id,send_json,0])
                    self.timer_list[dev_id]=t
                    t.start()
            self.sendRf(dev_id,send_json)
        else:
            logging.debug("Data received for non existent Dev")

    def callback(self,ch,method,properties,body):
        self.sendMssgResolv(body,properties)
    def sendRf(self,device,message):
        key=self.dev_mac[device]
        logging.debug(device+"-"+key)
        send_q[key].put(message)
    def updateDev(self,dev_id,status):
        #ToDo update Date of Device
        self.datab.updateStat(dev_id,status)
    def updateDevTime(self,dev_id,status):
        #ToDo update Date of Device
        self.datab.updateDateStat(dev_id,status,datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"))
    def scan(self):
        logging.debug("Scanning timeout")
        nearby_devices=bluetooth.discover_devices(lookup_names=True)
        logging.debug("found %d devices" %len(nearby_devices))
        for addr,name in nearby_devices:
            logging.debug(" %s - %s "% (addr,name))
            if name in self.find_devs:
                logging.debug("Attempting"+name+":"+addr+"port:"+str(self.port))
                try:
                    socket=bluetooth.BluetoothSocket(bluetooth.RFCOMM)
                    socket.connect((addr,port))
                    logging.debug('Connected to %s on port %s'%(name,port))
                    #port=port+1
                    socket.send('x')
                    socket.settimeout(0.1)
                    send_q[addr]=Queue.Queue(10)
                    receive_q[addr]=Queue.Queue(10)
                    bluet=self.BlueThread(name,socket,send_q[addr],receive_q[addr])
                    bluet.start()
                    threads[addr]=bluet
                except bluetooth.BluetoothError as error:
                    logging.error(error)
        if not exitFlag:
            mainTimer=Timer(300,self.scan)
            self.timer_list["main"]=mainTimer
            mainTimer.start()
```


Fig. B.4 Bluetooth Driver Snippet

Appendix C

Example Deployment File

The deployment file presented in Fig. C.1 is used to deploy applications and to save run-time parameters from optimisation tasks. This file shows how the resources, apps, gateways, clusters and the Fog are saved and what metadata each of them contains and how they are linked. The file is encoded in JSON and this snippet has sections cut out and removed so it will fit in a page. Examples of the full deployment files can be seen id the code base presented in the previous section.

```
{"Connections": {
        "1": {"Resources": {},
              "Apps": {
                "3": 2.6130548,
                "10": 3.1390471}},
        "2": {"Resources": {
                "1": 1.2181516,
                "2": 1.2181516
            },"Apps": {
                "6": 2.7414336}}},
    "date": "Test_javaWed Aug 22 10:31:13 BST 2018",
    "nodes": [{
            "id": 60001,
            "label": "TestGw0",
            "value": 47,
            "group": "server"},{
            "color": {
                "border": "black",
                "background": "Blue"},
            "shape": "box",
            "label": "TestAppC1_Nr1",
            "id": 1,
            "value": 18},{
            "image": "computer-microprocessor.png",
            "shape": "image",
            "label": "DeviceLoc_App2GW:3_0",
            "id": 10001,
            "value": 50},],
    "Applications": {
        "1": {
            "Type": "Testing_App7",
            "Load": 18.786345,
            "Utility": 0.92834944,
            "Constraints": {
                "Reliability": 1.4E-45,
                "Delay": 3.4028235E38},
            "TotalMsgCount": 5.752102,
            "UWeights": {
                "Constraints": 0.0,
                "Reliability": 0.33,
                "Delay": 1.0},
            "Delays": 146.47209,
            "Reliability": 0.5292551,
            "ConstViolations": 0.0,
            "Requirements": [
                "Capability C"],
            "Name": "TestAppC1_Nr1",
            "UnitLoad": 2.8}},
    "SystemData": {
        "Utility": 7.2170877,
        "Reliability": 4.5191536,
        "Delay": 1107.6079,
        "Name": "Test_java"},
    "Gateways": {
        "1": {
            "LIdle": 3.68,
            "Type": "Basic",
            "TotLoad": 59.540775,
            "TotMsgRate": 6.3332405,
            "Capabilities": [
                "Capability F",
                "Capability E",
                "Capability D",
                "Capability C",
                "Capability B",
                "Capability G",
                "Capability A"],
            "Resources": {
                "3": {
                    "Type": "Device",
                    "GatewayName": "TestGw0",
                    "GatewayID": 1,
                    "MsgRate": 3.4727967,
                    "Name": "DeviceLoc_App3GW:1_0"}},
            "BaseLoad": 8.574512,
            "Apps": {
                "1": {
                    "Type": "Testing_App7",
                    "Load": 18.786345,
                    "Utility": 0.92834944,
                    "Constraints": {
                        "Reliability": 1.4E-45,
                        "Delay": 3.4028235E38},
                    "TotalMsgCount": 5.752102,
                    "UWeights": {
                        "Constraints": 0.0,
                        "Reliability": 0.33,
                        "Delay": 1.0},
                    "Delays": 146.47209,
                    "Reliability": 0.5292551,
                    "ConstViolations": 0.0,
                    "Requirements": [
                        "Capability C"],
                    "Name": "TestAppC1_Nr1",
                    "UnitLoad": 2.8}},
            "Name": "TestGw0",
            "PerfCoef": 0.9445816}},
    "Latencies": {
        "1": {
            "2": 11.608755,
            "3": 30.405617}},
    "Clusters": {
        "1": {
            "Utility": 6.4431415,
            "Gateways": {
                "1": {
                    "Load": 80.81052,
                    "Share": 96.32,
                    "Name": "TestGw0"}},
            "Reliability": 4.037528,
            "ConstViolations": 4.0,
            "Apps": {
                "1": "TestAppC1_Nr1",
                "2": "TestAppC1_Nr2"},
            "Delay": 1036.3937,
            "Name": "Cluster_1"}},
    "edges": [{
            "color": {
                "color": "Blue"},
            "from": 1,
            "to": 60001}],
    "Resources": {
        "1": {
            "Type": "Device",
            "GatewayName": "TestGw2",
            "GatewayID": 3,
            "MsgRate": 1.2181516,
            "Name": "DeviceLoc_App2GW:3_0"}},
    "type": "virtual",
    "Name": "Test java"}
```


Fig. C.1 Example JSON Deployment File Snippet

Appendix D

Physical Backbone of the System

The physical backbone of the system supported all the tests and message management for the physical tests as well as provided the computational backbone for the Method Evaluation tests.The setup of the physical cluster that was used to support this can be seen in Fig. D.1. The software stack that was deployed to run the openstack cloud, spark cluster and other components can be seen in Fig. D.2. The Spark deployment can be seen in Fig. D.3. The devices that were used for testing and deployment can be seen in Fig. D.4.

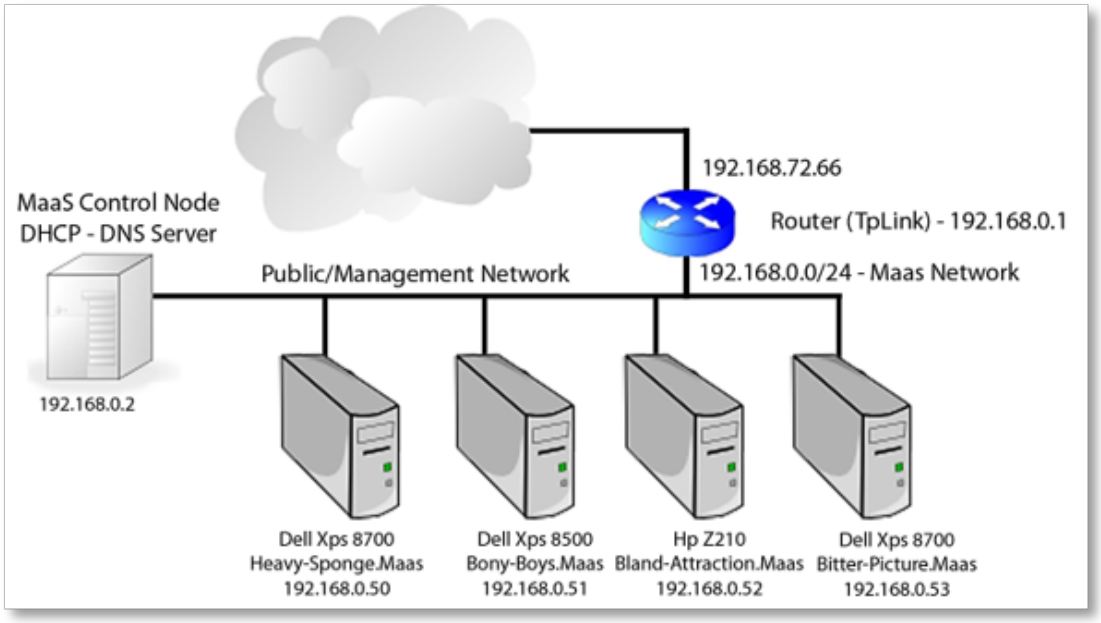

Fig. D.1 Physical Cluster

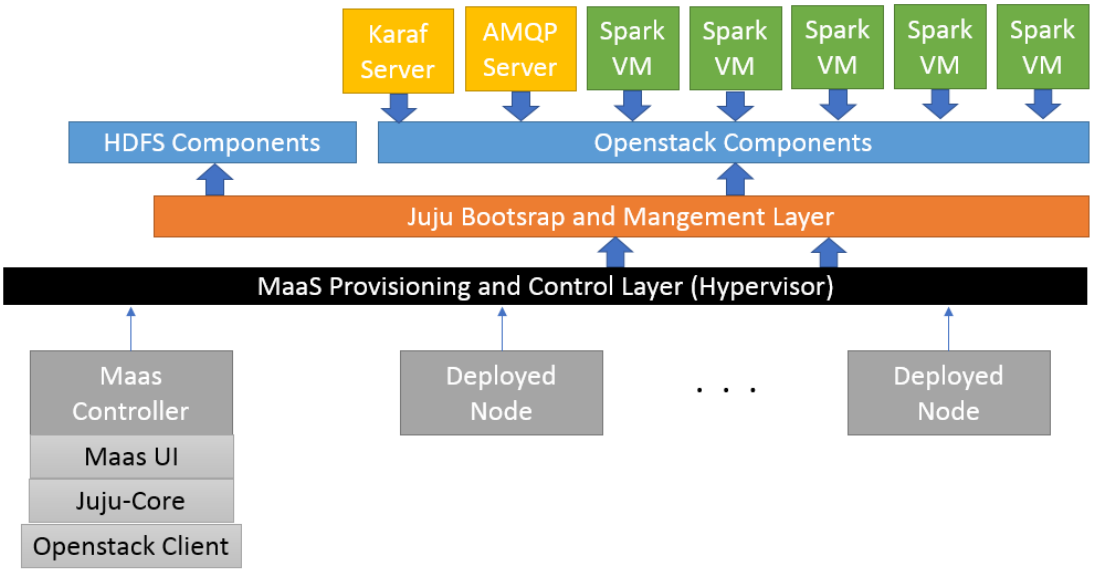

Fig. D.2 Software Stack

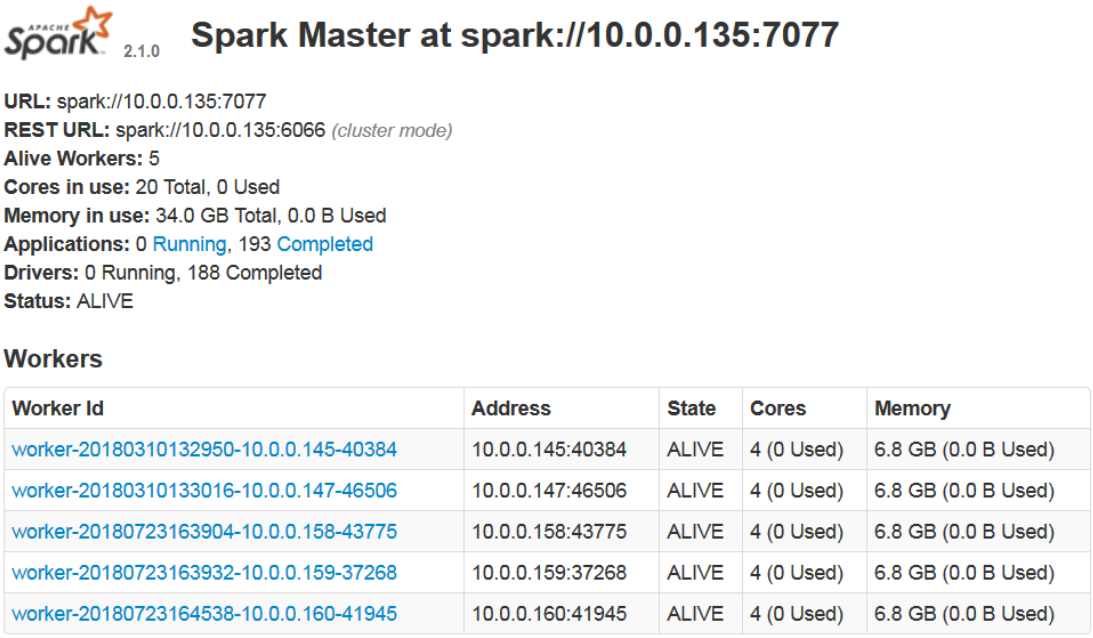

Fig. D.3 Spark Deployment

The Devices from Fig. D.4 a,b,c and g are Raspberry pi 2 System on Chip Fog Nodes that have attached varying sensor that allow them to interact with their surroundings. Device a. has a relay, a temperature and humidity sensor and an RF24 wireless transmitter for communicating with peripheral devices. The Raspberry pi from b. is designed to be used with video surveillance, as it has attached a motion sensor and a Video Camera. The device c.

has attached the temperature and humidity controller together with the RF24 module as well as a proximity sensor. The node in g. is designed to be a communication hub, as it can send and receive messages in all the testes technologies and has a higher range RF24 device. The device d. is a standard RF52 Thingy node that has a varying set of environmental sensors and communication devices but is used to tests the Low-Power Bluetooth connections with the Raspberry Pi. The XBee enabled Arduino board from e. has a light and temperature sensor for environmental monitoring attached to it. The AtTiny device from f. is designed to be a simple 3.3Volt powered monitoring device.

Some materials have been removed from this thesis due to Third Party Copyright. Pages where material has been removed are clearly marked in the electronic version. The unabridged version of the thesis can be viewed at the Lanchester Library, Coventry University.

Fig. D.4 Physical Devices and Nodes

Appendix E

Optimisation Run-time Log Example

A snippet of the output log from the performance validation tests can be seen in Fig. E.1. Here the main section of the tests are shown with the iterations excluded as they take up too much space.These exclusions are marked by ... in the text and could mean missing repeated lines or method iterations.

This tests contains the outputs of all 5 methods and the initial generations as well. The results are for a 320 application deployment for the Delay scenario, but might not be the one presented in the evaluation section.

```
%Driver Log :  stdout log page for driver-20180814133134-0275
Args:[Perf] Size:2 sceType:1 meType:0
Starting Performance Test.
---------------------------
-----Generating new Fog-----
---------------------------
Parameters - AppCount: 320 Gw Count's Ext: 0.1 Load: 60.0 Latency: 8.97,30.897Ext Lat: 37.37,87.89 Fog Type: Delay
Tot Res Required Count: 5883.5796  Gw Count: 117 ResProvided: 13461.3955 at Rate: 2.28796 App Cnt: 320 Utility: 808.73474
---------------------------
----- A. Global GA Stuff -----
---------------------------
GA Global GwCount:117 AppCnt:320 Size:72 Gens:1070
The best of the Population 340 is: 800.79224 At: 960.172
...
The best of the Population 5275 is: 822.05914 At: 12142.278
Best of 6398  Population was at: 5328 is: 822.14844 At: 14838.124
----------------------------------------
----- B. Distance Clustering Deployment -----
----------------------------------------
----- Clustering -----
Cluster 1 Apps: [130, ... 110]
Cluster 2 Apps: [128, ... 255]
Cluster 3 Apps: [256, ... 254]
Cluster 4 Apps: [129, ... 250]
 -> GA Cluster 1 GwCount:55 AppCnt:80 Size:54 Gens:550
Ga Failed at pop 5000 !
{130=12, 4=11, 5=96, 133=24, ... 110=36}
GW: 12 Load: 74.59566 MaxLoad: 47.067543
...
GW: 96 Load: 112.97151 MaxLoad: 39.856384
Final Resp:0
Utility:205.68835
Distance Clustering Failed
------------------------------------------------------------
----- C. Sample Weighted Distance Clustering Optimization -----
------------------------------------------------------------
-> SampleClustering
 -> GA Cluster 1 GwCount:13 AppCnt:59 Size:52 Gens:126
The best of the Population 1205 is: 131.76009
Sample Clustering finished in :111.284
Sampling Best Util:131.76007080078125 with solution: {4=107, 132=110, ... 124=113, 253=112}
Sampling Finished in :111.293
----------------------------------------
New Iteration of Training Algorithm Started
----------------------------------------
Apps Correlations: {Constraints=0.0, RequirementSim=0.009922318063188777, ResourceShare=0.036468574365358955,
UtilityWeights=0.0, MessageRate=-0.007976248954193159, Distance=-0.003333345863698447, UnitLoad=0.024539101908489496}
Gws Correlations: {Capabilities=0.0, SharedRes=0.010033092253716744, PerfToULoad=0.12285470794435353,
BaseLoad=-0.03805291556295542, CapToULoad=0.07774400188732135}
 -> Sorting Correlation Results - Not Dir Stop -
Clustering Parameters: Count:1 FailSteps:0 ProcLim[app/gw]:0.2/0.05 App-Penalties:{} Gw-Penalties:{} ----------
Weight Apps:{RequirementSim=0.12574820000702708, ResourceShare=0.4621760312521691, MessageRate=-0.10108514385552601,
UnitLoad=0.3109906248852778}Weight Gws:{SharedRes=0.040344627320059334, PerfToULoad=0.49401792400478906,
BaseLoad=-0.15301670292729525, CapToULoad=0.31262074574785637}
-> Clustering
Eps Vals:[-0.06666663905476547, 0.4672202900299937, 0.05338869290847592]
Eps Search Results - Best Eps:0.2536655 BestValid: 12.25
Cluster 1 Size:42 Apps: [320, ... 187]
...
Cluster 8 Size:21 Apps: [128, ... 157]
Unallocated Apps: []
 -> GA Cluster 1 GwCount:27 AppCnt:42 Size:51 Gens:467
The best of the Population 1968 is: 110.29555
...
 -> GA Cluster 8 GwCount:15 AppCnt:21 Size:49 Gens:422
The best of the Population 1534 is: 54.396114
Fog Utility: 820.6405
Direction Clustering Done in :873.417
----------------------------------------
New Iteration of Training Algorithm Started
----------------------------------------
Apps Correlations: {Constraints=0.0, RequirementSim=-0.005347581101487097, ResourceShare=0.0020892691508377077,
UtilityWeights=0.0, MessageRate=0.0024588056933909507, Distance=-0.01297322160741293, UnitLoad=0.0033775152902921135}
Gws Correlations: {Capabilities=0.0, SharedRes=0.016669262544068475, PerfToULoad=-0.018756340143875377,
BaseLoad=-0.0030906981605134613, CapToULoad=-0.01379416814175487}
 -> Sorting Correlation Results - Not Dir Stop -
Clustering Parameters: Count:2 FailSteps:0 ProcLim[app/gw]:0.2/0.05 App-Penalties:{} Gw-Penalties:{} ----------
Weight Apps:{RequirementSim=-0.24645141165717063, Distance=-0.5978906571419933, UnitLoad=0.1556579312008361}Weight
```

```
Gws:{SharedRes=0.3186601624081684, PerfToULoad=-0.358558057420301, BaseLoad=-0.059083740218268004,
CapToULoad=-0.26369803995326246}
-> Clustering
Eps Vals:[-2.6224482468731116, -2.220446049250313E-16, 0.26224482468731114]
Eps Search Results - Best Eps:-1.3112241 BestValid: 4.777777777777778
Cluster 1 Size:24 Apps: [1, ... 249]
...
Cluster 12 Size:26 Apps: [257, 194, 258, 136, 200, 138, 267, 81, 275, 84, 212, 21, 152, 153, 154, 219, 284, 286, 225, 103,
296, 169, 299, 51, 311, 313]
Unallocated Apps: []
 -> GA Cluster 1 GwCount:9 AppCnt:24 Size:49 Gens:428
The best of the Population 843 is: 63.25459
 ...
 -> GA Cluster 12 GwCount:10 AppCnt:26 Size:49 Gens:433
The best of the Population 494 is: 67.35516
Fog Utility: 826.7943
Direction Clustering Done in :1144.421
...
-----------------------------------------
New Iteration of Training Algorithm Started
-----------------------------------------
Apps Correlations: {Constraints=0.0, RequirementSim=5.459132045650533E-4, ResourceShare=0.04517603949826375,
UtilityWeights=0.0, MessageRate=-3.9374402624911135E-4, Distance=-0.03531882268272359, UnitLoad=-4.510048126500031E-4}
Gws Correlations: {Capabilities=0.0, SharedRes=0.02152665962796101, PerfToULoad=-0.0198794274493012,
BaseLoad=-6.088063042454969E-4, CapToULoad=-0.019385437365909697}
 -> Sorting Correlation Results - Dir Stop -  - Worse Util
Underfitted App Solution, Solving...
Underfitted Gw Solution, Solving...
Clustering Parameters: Count:7 FailSteps:2 ProcLim[app/gw]:0.6773759999999999/0.16934399999999997 App-Penalties:{}
Gw-Penalties:{} ----------
Weight Apps:{ResourceShare=1.0}Weight Gws:{SharedRes=0.25765659901443466, PerfToULoad=-0.39478359619918335,
CapToULoad=-0.3475598047863821}
-> Clustering
Eps Vals:[-0.1, 0.21000000000000002, 0.031000000000000007]
Eps Search Results - Best Eps:-0.069000006 BestValid: 45.5
Cluster 1 Size:162 Apps: [256, ... 204]
Cluster 4 Size:30 Apps: [194, ... 319]
Unallocated Apps: []
 -> GA Cluster 1 GwCount:57 AppCnt:162 Size:60 Gens:728
The best of the Population 3676 is: 416.03424
 ...
 -> GA Cluster 4 GwCount:11 AppCnt:30 Size:50 Gens:441
The best of the Population 605 is: 77.55198
Fog Utility: 825.8747
Direction Clustering Done in :2406.059
Results:
Clust[Count:1 FailSteps:0 ProcLim[app/gw]:0.2/0.05 App-Penalties:{} Gw-Penalties:{} Time: 873.417] = 820.6405
Clust[Count:2 FailSteps:0 ProcLim[app/gw]:0.2/0.05 App-Penalties:{} Gw-Penalties:{} Time: 1144.421] = 826.7943
Clust[Count:3 FailSteps:1 ProcLim[app/gw]:0.24/0.06 App-Penalties:{} Gw-Penalties:{} Time: 1912.109] = 827.0376
Clust[Count:4 FailSteps:1 ProcLim[app/gw]:0.288/0.072 App-Penalties:{} Gw-Penalties:{} Time: 1248.964] = 825.09515
Clust[Count:5 FailSteps:2 ProcLim[app/gw]:0.40319999999999995/0.10079999999999999 App-Penalties:{} Gw-Penalties:{} Time:
1706.234] = 827.6831
Clust[Count:6 FailSteps:1 ProcLim[app/gw]:0.48383999999999994/0.12095999999999998 App-Penalties:{} Gw-Penalties:{} Time:
1329.65] = 826.39886
Clust[Count:7 FailSteps:2 ProcLim[app/gw]:0.6773759999999999/0.16934399999999997 App-Penalties:{} Gw-Penalties:{} Time:
2406.059] = 825.8747
Method Finished in :10732.306
-------------------------------------------------------------------------
----- D. Initial Weights Weighted Distance Clustering Optimization -----
-------------------------------------------------------------------------
-----------------------------------------
New Iteration of Training Algorithm Started
-----------------------------------------
 -> Sorting Correlation Results - Not Dir Stop -
Clustering Parameters: Count:0 FailSteps:0 ProcLim[app/gw]:0.2/0.05 App-Penalties:{} Gw-Penalties:{} ----------
Weight Apps:{}Weight Gws:{}
-> Clustering
Eps Vals:[4.779385345431594, 8.687495095131421, 0.39081097496998274]
Eps Search Results - Best Eps:6.7334404 BestValid: 5.166666666666667
Cluster 1 Size:22 Apps: [32, ... 45]
Cluster 12 Size:27 Apps: [69, ... 179]
Unallocated Apps: []
 -> GA Cluster 1 GwCount:12 AppCnt:22 Size:49 Gens:424
The best of the Population 877 is: 54.259914
 ...
 -> GA Cluster 12 GwCount:14 AppCnt:27 Size:49 Gens:435
The best of the Population 2170 is: 68.890045
Fog Utility: 822.46436
```

```
D:\Doktori\Thesis\Thesis Draft\Appendix\Log.txt                                  Wednesday, August 22, 2018 12:26 PM
Direction Clustering Done in :987.234
...
-----------------------------------------
New Iteration of Training Algorithm Started
-----------------------------------------
Apps Correlations: {Constraints=0.0, RequirementSim=0.0046098243147765635, ResourceShare=0.059628816649848336,
UtilityWeights=0.0, MessageRate=-0.013079744769438892, Distance=-0.029517770385124122, UnitLoad=-0.005124293261328776}
Gws Correlations: {Capabilities=0.0, SharedRes=0.031574343489963276, PerfToULoad=-0.023927695874447683,
BaseLoad=0.0039904809171321855, CapToULoad=-0.020199237714567705}
 -> Sorting Correlation Results - Dir Stop -  - Worse Util
Underfitted App Solution, Solving...
Underfitted Gw Solution, Solving...
Clustering Parameters: Count:4 FailSteps:2 ProcLim[app/gw]:0.40319999999999995/0.10079999999999999 App-Penalties:{}
Gw-Penalties:{} ----------
Weight Apps:{ResourceShare=0.4827881117929911, Distance=-0.5172118882070088}Weight Gws:{SharedRes=0.6318911023239071,
PerfToULoad=-0.21571606599339305, CapToULoad=-0.15239283168269985}
-> Clustering
Eps Vals:[-2.275732308110839, -0.20688475528280353, 0.20688475528280356]
Eps Search Results - Best Eps:-1.4481932 BestValid: 14.081632653061225
Cluster 1 Size:39 Apps: [128, ... 131]
Cluster 7 Size:42 Apps: [320, ... 63]
Unallocated Apps: []
 -> GA Cluster 1 GwCount:16 AppCnt:39 Size:50 Gens:461
The best of the Population 2448 is: 102.66226
 ...
 -> GA Cluster 7 GwCount:15 AppCnt:42 Size:51 Gens:467
The best of the Population 2366 is: 106.1328
Fog Utility: 826.51874
Direction Clustering Done in :1796.233
Results:
Clust[Count:0 FailSteps:0 ProcLim[app/gw]:0.2/0.05 App-Penalties:{} Gw-Penalties:{} Time: 987.234] = 822.46436
Clust[Count:1 FailSteps:0 ProcLim[app/gw]:0.2/0.05 App-Penalties:{} Gw-Penalties:{} Time: 1552.933] = 826.12396
Clust[Count:2 FailSteps:1 ProcLim[app/gw]:0.24/0.06 App-Penalties:{} Gw-Penalties:{} Time: 1111.917] = 827.0043
Clust[Count:3 FailSteps:1 ProcLim[app/gw]:0.288/0.072 App-Penalties:{} Gw-Penalties:{} Time: 1374.718] = 826.58
Clust[Count:4 FailSteps:2 ProcLim[app/gw]:0.40319999999999995/0.10079999999999999 App-Penalties:{} Gw-Penalties:{} Time:
1796.233] = 826.51874
Method Finished in :6823.156
----------------------------
----- B(0). Random Deployment -----
----------------------------
-> Clustering
Cluster 1 Apps: [1,... 5]
Cluster 12 Apps: [288, 289, 290, 293, 262, 295, 302, 307, 277, 278, 310, 312, 281, 313, 314, 316, 317, 286, 318]
Unallocated Apps: []
 -> GA Cluster 1 GwCount:10 AppCnt:29 Size:49 Gens:439
The best of the Population 1011 is: 75.047325
...
 -> GA Cluster 12 GwCount:8 AppCnt:19 Size:49 Gens:418
The best of the Population 880 is: 49.15943
Fog Utility: 826.2746
Finished Clusering Part in:443.6
Unalocated Apps: []
Total Elapsed Time:443.6
Utilities...
Init: 827.0043
Random: 828.2746
Dist: 0.0
Sample: 827.6831
GA: 822.14844
X.Init = [0.012 ... 6823.15 ];
X.Random = [0.0 0.171 0.18 443.594 ];
X.Dist = [0.0 21.328 21.331 ];
X.Sample = [0.0 ... 10732.299 ];
X.GA = [3.625 ... 14818.575 ];
Y.Init = [0.0 ... 827.0043 ];
Y.Random = [0.0 0.0 0.0 826.2746 ];
Y.Dist = [0.0 0.0 0.0 ];
Y.Sample = [0.0 ... 827.6831 ];
Y.GA = [0.0 ... 822.14844 ];
size = 3; type = 1;
XMat(size,type) = X;
YMat(size,type) = Y;

                                                                                -3-
```

Fig. E.1 Performance Test Log Example