UMEÅ UNIVERSITY

Resource Allocation for Mobile Edge Clouds

Amardeep Mehta

PhD Thesis

Department of Computing Science
Umeå University
Sweden
2018

Department of Computing Science
Umeå University
SE-901 87 Umeå
Sweden

The work is funded in part by the Swedish Research Council(VR) under contract number C0590801 for the Cloud Control project.

ISBN: 978-91-7601-925-2
ISSN: 0348-0542
UMINF: 18.10

Electronic version available at http://umu.diva-portal.org/
Printed by UmU Print Service, Umeå University
Umeå, Sweden 2018

Abstract

Recent advances in Internet technologies have led to the proliferation of new distributed applications in the transportation, healthcare, mining, security, and entertainment sectors. The emerging applications have characteristics such as being bandwidth-hungry, latency-critical, and applications with a user population contained within a limited geographical area, and require high availability, low jitter and security.

One way of addressing the challenges arising because of these emerging applications, is to move the computing capabilities closer to the end-users, at the logical edge of a network, in order to improve the performance, operating cost, and reliability of applications and services. These distributed new resources and software stacks, situated on the path between today's centralized data centers and devices in close proximity to the last mile network, are known as Mobile Edge Clouds (MECs). The distributed MECs provides new opportunities for the management of compute resources and the allocation of applications to those resources in order to minimize the overall cost of application deployment while satisfying end-user demands in terms of application performance.

However, these opportunities also present three significant challenges. The first challenge is *where* and *how much* computing resources to deploy along the path between today's centralized data centers and devices for cost-optimal operations. The second challenge is *where* and *how much* resources should be allocated to *which* applications to meet the applications' performance requirements while minimizing operational costs. The third challenge is *how* to provide a framework for application deployment on resource-constrained IoT devices in heterogeneous environments.

This thesis addresses the above challenges by proposing several models, algorithms, and simulation and software frameworks. In the first part, we investigate methods for early detection of short-lived and significant increase in demand for computing resources (also called spikes) which may cause significant degradation in the performance of a distributed application. We make use of adaptive signal processing techniques for early detection of spikes. We then consider trade-offs between parameters such as the time taken to detect a spike and the number of false spikes that are detected. In the second part, we study the resource planning problem where we study the cost benefits of adding new compute resources based on performance requirements for emerging applications. In the third part, we study the problem of allocating resources to applications by formulating as an optimization problem, where the objective is to minimize overall operational cost while meeting the performance targets of applications. We also propose a hierarchical scheduling framework and policies for allocating resources to applications based on performance metrics of both applications and compute resources. In the last part, we propose a framework, *Calvin Constrained*, for resource-constrained devices, which is an extension of the *Calvin* reference framework and supports a limited but essential subset of the features of the reference framework taking into account the limited memory and processing power of the resource-constrained IoT devices.

Sammanfattning

Dagens utveckling av olika Internetteknologier har gett upphov till nya typer av distribuerade applikationer inom områden som transport, hälso- och sjukvård, gruvdrift, säkerhet och underhållning. Några exempel är självkörande fordon, augmenterad verklighet, smarta hem och städer samt videoövervakning och sakernas Internet. Dessa typer av applikationer ger upphov till enorma mängder data. De kan även användas nästan uteslutande i vissa lokala områden, ställa höga krav på tillgänglighet och säkerhet samt kräva mycket bandbredd, låg latens och minimalt med jitter. En ansats att möta dessa krav är att flytta beräkningskapacitet från centraliserade datacenter till yttersta kanten av nätverket. Detta paradigm med decentraliserade beräkningsresurser som komplement till dagens datacenter benämns Mobile Edge Clouds (MECs). Denna distribuerade infrastruktur ger möjligheter till holistisk hantering av resurser och applikationer för att uppfylla applikationers prestandakrav och samtidigt minimera infrastrukturkostnaden, men ger även upphov till en rad utmaningar. Den första utmaningen är hur mycket beräkningskapacitet som ska placeras i nätverket mellan datacentren och slutanvändarna för att få kostnadsoptimal drift, och var denna kapacitet ska placeras. Den andra, relaterade, frågeställningen är var och hur mycket kapacitet som ska allokeras till varje applikation, och var, för att tillgodose prestandakrav och minimera kostnader. Den tredje utmaningen är hur ett ramverk kan designas för att möjliggöra en standardiserad utvecklingsprocess för resursbegränsade applikationer som kommer att köras i heterogena miljöer.

Denna avhandling adresserar dessa utmaningar genom att introducera modeller, algoritmer och ramverk. I första delen undersöks metoder för tidig detektion av plötsliga förändringar i belastningsmönster (antal anrop till applikationen), så kallade spikar, vilka kan ge upphov till försämrad prestanda för en distribuerad applikation. Adaptiva metoder för signalbehandling utvärderas efter vissa kriterier, såsom hur tidigt en spik upptäcks och hur ofta metoderna felaktigt detekterar spikar. I den andra delen studeras resursplaneringsproblemet genom att utvärdera fördelar och kostnader med installation av ytterligare beräkningskapacitet i MECs. Utvärderingen baseras på krav och parametrar för tänkbara framtida distribuerade applikationer och i kontext av en optimal algoritm för allokering av resurser till applikationer. I tredje delen undersökts resursallokering för applikationer, vilket formuleras som ett optimeringsproblem där målet är att minera totala driftkostnaden och samtidigt uppfylla alla prestandakrav. En hierarkisk schedulingsmetod baserad på applikationers prestandamått introduceras och utvärderas med hjälp av simuleringar. I sista delen föreslås en utökning Calvin-systemet i form av ett reducerat ramverk med vissa nyckelfunktioner för att kunna köra Calvin-applikationer på system med begränsat minne och beräkningskraft, vilka är vanliga förekommande i MECs.

Acknowledgments

The PhD thesis has been one of the most significant academic challenges I have faced in my life. During the process, I have got assistance from many people around me. In particular, my deepest gratitude goes to my supervisor **Erik Elmroth** and co-supervisor **Johan Tordsson**. Without their guidance, this work could not have been possible.

I would like to thank my co-authors – **William Tärneberg, Cristian Klein, Maria Kihl, Ewnetu Bayuh Lakew, Eddie Wadbro, Jonas Dürango, Fredrik Svensson, Harald Gustafsson, Johan Eker, Ahmed Ali-Eldin,** and **Chanh Nguyen** – for the valuable discussions we have had during our research collaborations. I would also like to thank William for his base graphics image that I modified and used in this thesis. I appreciate all my current and former colleagues in the distributed systems group – **Abel, Daniel, Fransisco, Gonzalo, Jakub, Lars, Mina, Muyi, Monowar, Peter, Petter, P–O, Thang, Tobias**, and **Viali** – for their help, support, and providing a great atmosphere during my work.

I would like to thank people at the department, especially **Anne-Lie Persson**, **Yvonne Löwstedt**, **Carina Gustafsson**, and **Lennart Edblom** for handling administrative aspects and being great company during my studies. I would also like to thank our system administrators, **Mattias**, **Matts**, **Bertil**, and **Tomas** for helping me out with technical issues.

I would like to thank colleagues from Ericsson Research for providing the wonderful environment and sharing their knowledge during my past two internships at Stockholm and Lund.

I would also thank all my friends, especially **Adrian**, **Niclas**, **Mukund**, **Rajesh**, **Shreejit** and **Juan**, and relatives, for their love, support, and encouragement.

My deep and sincere gratitude goes to my family for their love and continuous support throughput my studies. I would like to thank my wife **Omi** for her love, understanding, and encouragement, and my beloved daughter **Anisha** for always being my stress-buster. I am forever indebted to my dear parents, **Siya Ram Mehta** and **Bimla Devi**, for nurturing my learning, supporting my dreams, and giving me opportunities that made me who I am. I would also like to thank my mother- and father-in-law for their love, support, and encouragement.

Preface

This thesis consists of an introductory chapter and the following papers:

I. Amardeep Mehta, Jonas Dürango, Johan Tordsson, and Erik Elmroth, *Online Spike Detection in Cloud Workloads*, In Proceedings of the IEEE International Conference on Cloud Engineering (IC2E), pp. 446-451, IEEE, 2015.

II. William Tärneberg, Amardeep Mehta, Eddie Wadbro, Johan Tordsson, Johan Eker, Maria Kihl, and Erik Elmroth, *Dynamic Application Placement in the Mobile Cloud Network*, Future Generation Computer Systems, pp. 163-177, Elsevier, 2017.

III. Amardeep Mehta, William Tärneberg, Cristian Klein, Johan Tordsson, Maria Kihl, and Erik Elmroth, *How Beneficial are Intermediate Layer Data Centers in Mobile Edge Networks?*, In Proceedings of the IEEE 1st International Workshop on Foundations and Applications of Self-*Systems, pp. 222-229, IEEE, 2016.

IV. Amardeep Mehta, Rami Baddour, Fredrik Svensson, Harald Gustafsson, and Erik Elmroth, *Calvin Constrained – A Framework for IoT Applications in Heterogeneous Environments*, In Proceedings of the IEEE 37th International Conference on Distributed Computing Systems (ICDCS), pp. 1063-1073, IEEE, 2017.

V. Amardeep Mehta and Erik Elmroth, *Distributed Cost-Optimized Placement for Latency-Critical Applications in Heterogeneous Environments*, In Proceedings of the IEEE 15th International Conference on Autonomic Computing (ICAC), to appear, 2018.

VI. Amardeep Mehta, Ewnetu Bayuh Lakew, Johan Tordsson, and Erik Elmroth, *Utility-based Allocation of Industrial IoT Applications in Mobile Edge Clouds*, Submitted for publication, 2018.

In addition to the papers included in the thesis, following papers have been produced during the studies:

- Ahmed Ali-Eldin, Ali Rezaie, Amardeep Mehta, Stanislav Razroev, Sara Sjöstedt de Luna, Oleg Seleznjev, Johan Tordsson, and Erik Elmroth, *How Will Your Workload Look Like in 6 Years? Analyzing Wikimedia's Workload*, In Proceedings of the 2014 IEEE International Conference on Cloud Engineering (IC2E), pp. 349-354, IEEE, 2014.

- William Tärneberg, Amardeep Mehta, Johan Tordsson, Maria Kihl, and Erik Elmroth, *Resource Management Challenges for the Infinite Cloud*, In 10th International Workshop on Feedback Computing at CPSWeek, pp. 1-4, 2015.

- William Tärneberg, Alessandro Vittorio Papadopoulos, Amardeep Mehta, Johan Tordsson, and Maria Kihl, *Distributed Approach to the Holistic Resource Management of a Mobile Cloud Network*, In IEEE 1st International Conference on Fog and Edge Computing (ICFEC), pp. 51-60, IEEE, 2017.

- Chanh Nguyen, Amardeep Mehta, Cristian Klein, and Erik Elmroth, *Why Cloud Applications Are not Ready for the Edge (yet)*, Submitted for publication, 2018.

Introduction

1 Introduction

1.1 Background and Research Motivation

Recent advances in the cost, performance, and energy efficiency of IoT devices, network technologies (such as 5G), and distributed computing architectures have led to the explosive growth of the Internet and mobile connectivity, in turn leading to new distributed applications in areas such as transportation, healthcare, mining, entertainment, and security, such as automated vehicles, augmented reality, cloud robotics, smart homes and cities, video surveillance and streaming, and Internet of Things (IoT) applications. This has led to an unprecedented growth in data as well as increasing the importance of latency and regulation in handling and managing data [1, 2]. The new distributed applications have characteristics which may be bandwidth-hungry (video surveillance, video conferencing, traffic monitoring), latency-critical (automated vehicles, robotic surgery, safety), and may cause spikes in activity at particular places or times (sporting events). Applications may also require high availability, low jitter and security.

Large-scale deployment of Internet of Things (IoTs) and Industrial Internet of Things (IIoTs) devices is expected to play a big role in the development of smart cities, which will generate large volumes of aggregated cellular data that may choke the network. On the other hand, devices such as sensors on the power grid or on oil pipelines may host latency-sensitive applications that will require low latency in order to ensure that mission-critical data is transmitted and processed in a timely manner so that potential damage to people, property and environment can be averted [1]. Online video games on consoles such Xbox Live, where reaction times are in milliseconds, have become very popular recently. Such games may be hosted on a distant Data Center (DC), so the presence of latency and jitter could have a significant effect on gamers' experience and dramatically reduce their interest in the games.

Virtual Reality (VR), Augmented Reality (AR) and other state-of-the-art human-computer interaction applications require low latency and rapid processing for complex rendering algorithms as well as large volumes of data that may need to be transfered between a user and a DC hosting the applications.

Autonomous Vehicles (AVs) are the new trend in the transportation industries as their benefits include improved safety, enhanced efficiency and reduced manual labor for millions of people. Such vehicles will require high-performance compute engines to simulate hundreds of times every second all the cognitive functions that a human performs while driving. It is impractical and prohibitively expensive to expect on-board systems to perform all these functions [1]. However, offloading these functions to a DC will also require low latency to ensure operational safety at an efficient speed for the AVs.

There has been surge of live streaming applications, such as SnapChat, Facebook live, YouTube live, due to the proliferation of High Definition (HD) video cameras on smartphones. The volume of data created by these applications is creating significant pressure on upstream cellular networks. Similarly, video surveillance applications will

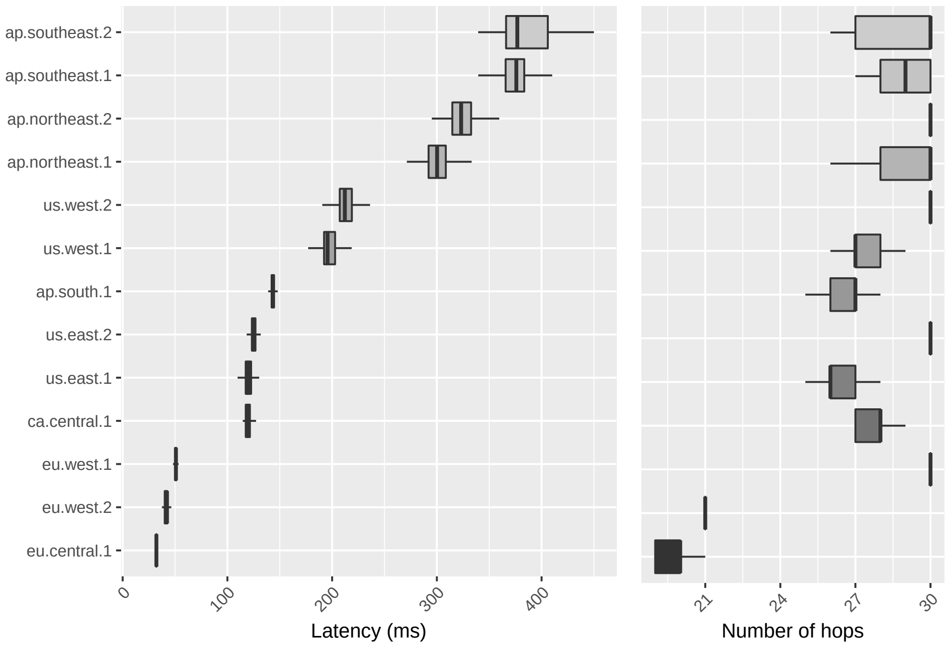

Figure 1: Round Trip Time (RTT) and number of hops for selected lambda Amazon Web Services (AWS) endpoints [4] from Umeå University.

require high-performance compute resources to run Artificial Intelligence (AI) and Machine Learning (ML) technologies in order to identify people and alert human operators in real time. These applications may generate gigabytes or even terabytes of data per second.

All in all, there is a need for a compute infrastructure that can begin to address the challenges posed by these emerging applications. Currently, Centralized Data Centers (CDCs) or cloud DCs can provide high-volume compute and storage resources. They benefit from the statistical multiplexing of the applications' compute resource requirements as well as lower cost due to economies of scale [3], but are unable to cope with the Quality of Service (QoS) requirements of many future Information and Communications Technology (ICT) applications due to long-distance network connectivity. Latency for a CDC could be at least 20-40 ms as shown in the Figure 1. The average number of hops to reach a CDC could be as much as 30, and, the greater the number of hops, the greater the probability of packet loss and network congestion.

Today's telecom networks are not even expected to handle the enormous and rapidly varying capacity demands that will arise in the near future. One of the challenges associated with realizing the full potential of IoT and IIoT applications is how to handle the network traffic between end-users and application-hosting nodes while

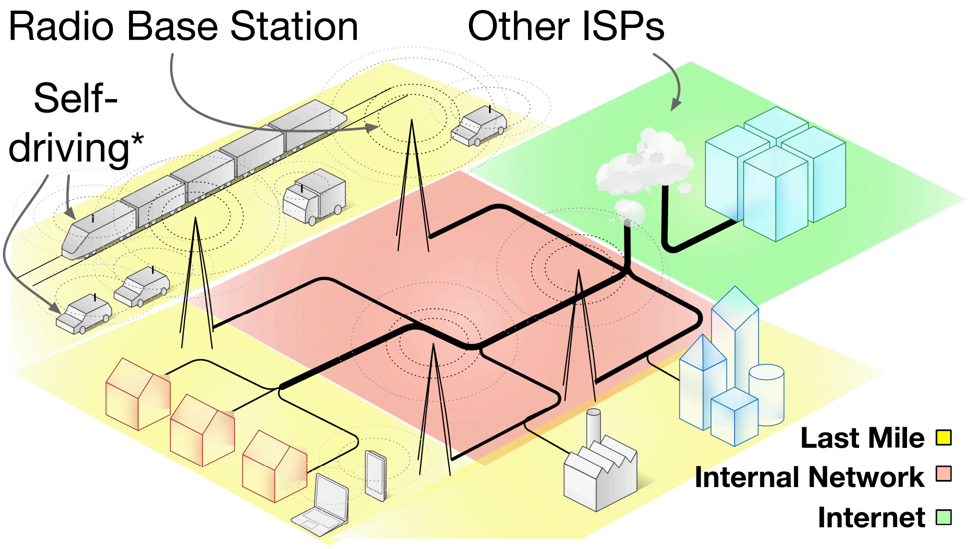

Figure 2: Three stage traversal for a cloud based application traffic through an ISP's network.

minimizing the operating cost of the infrastructure and meeting the QoS requirements of end-users, such as latency and/or throughput.

The traffic associated with ICT applications in the future may have to traverse three different forms of (inter)network if they are hosted by a CDC, as shown in Figure 2. The first stage – the *last-mile* – is the link between the end-user and the edge network of an Internet Service Provider (ISP); that is, the point at which an ISP begins to route user traffic to its intended destination. Congestion in the last-mile has been mitigated by recent enhancements in bandwidth capabilities from new broadband and radio access technologies, such as 5G. The second stage – the *internal network* – runs from the edge network up to the point where the Internet Service Provider (ISP) hands off the aggregated application traffic to various network points of another provider. Congestion can occur in an ISP's internal network when the aggregated demand exceeds the network capacity at some point in the network. The third stage – the *Internet* – is where off-premises or cloud DCs are situated. The ISP's internal network may not be able to meet the challenges of future, bandwidth-hungry, Internet-based applications due to expensive backhauling and increasing wired network congestion [5]. Furthermore, the increased latency, due to the congested internal network, may cause poor performance for the distributed applications [6].

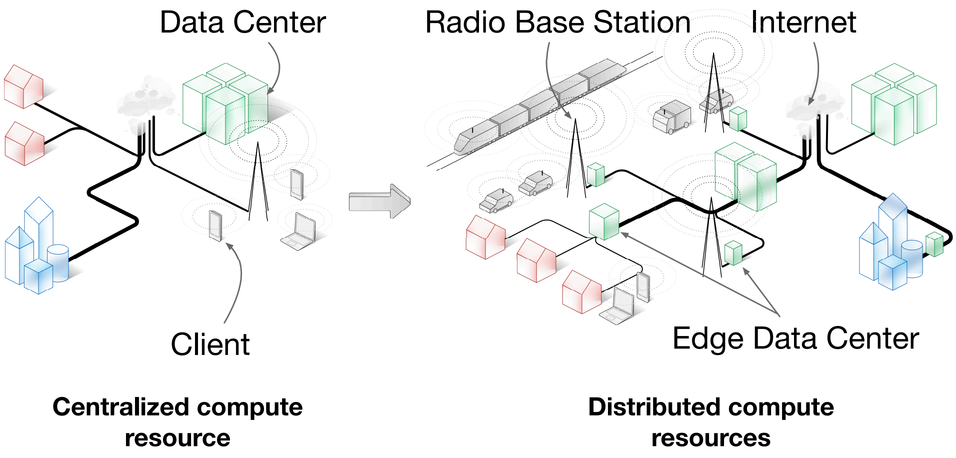

Figure 3: Centralized vs. distributed compute resource infrastructure [11].

1.2 Research Problems and Objectives

One way to address the challenges presented by emerging IoT applications, is to move the computations closer to end-users – that is, towards the ISP's edge network – in order to reduce transmission costs, decrease network latency and jitter, increase reliability, and avoid network congestion. A key idea is to create a unified ICT infrastructure using existing large-scale distributed cloud infrastructures and augmenting them with compute capacities at intermediary nodes, such as radio base stations at the ISP's edge network and inside its internal network. The unified infrastructure at the network edge, called Mobile Edge Clouds (MECs), as shown in Figure 3, can host applications closer to the end-users, thereby alleviating congestion problems and meeting the performance expectations of end-users [7, 8, 9, 10]. Several different names for similar concepts are used in the literature, such as Mobile Edge Networks [7], Mobile Cloud Networks [11, 12], Mobile Edge computing [13, 14], Mobile Micro-Clouds [15], Fog computing [16, 17, 18], Mobile cloud computing [19] and Telco-cloud [20].

In order to cope with the expected exponential growth in demand for compute resources from emerging applications, MECs must address four main challenges: (1) Understanding the resource demands of emerging applications, (2) Adding capacity at the network edge optimally, (3) Keeping capacity optimized, (4) Handling the heterogeneity of resources. The following problems are studied in this thesis:

- **How to model the resource demands and QoS requirements of applications?**

  Understanding the resource demands and QoS requirements of applications, is crucial when designing resource allocation algorithms capable of meeting the demands of emerging MEC applications [8]. The relevant parameters of an application need to be modeled to understand the underlying system dynamics, such as cost and performance dynamics [7]. Sensitivity analysis also needs to be performed in order to identify and focus on the parameters that have a significant impact on the algorithms' results [7].

The resource demands of applications may vary over time in a predictable manner, or they may experience *spikes* or *bursts* [21, 22]. These spikes need to be detected as early as possible so that appropriate resources can be allocated and thereby prevent application slowdown or failure [21]. These resource demand models can also be used to generate test cases that can be used as inputs to resource allocation algorithms for MECs [11].

- **How to plan capacity and infrastructure for MECs?**

  Planning and building a new DC or adding capacity to a DC may take months or years, so it is important to perform a cost analysis to understand which applications will benefit by building or enhancing DCs between the end-users and the CDC. Operational and capital costs for a DC need to be considered, as they vary with the capacity of the DC due to economies of scale [3]. The trade-off between cheap compute cost but higher bandwidth cost at distant DCs and expensive compute cost but lower bandwidth cost at an MEC leads to an essential question: Where should an MEC be located and how much compute capacity does it need to be provided with?

- **How to optimally allocate resources to applications in MECs?**

  The main factors that affect the resource allocation problem are: sensitivity of applications' request with respect to resource types, space and time variation in their demand for resources, QoS requirements, and the capabilities of the available compute resources. For example, an application's requests (for resources) may be modeled by various parameters, such as arrival rate, service time and data size, and each of these parameters may be modeled using a random variable from an appropriate distribution. Similarly, the location of an end-user running an application may be modeled by a suitable probability distribution [11].

  The performance of applications may have to satisfy Key Performance Indicators (KPIs) specified by the end-users. For example, some applications may require that the average Round Trip Time (RTT) is below a certain threshold [11], whereas others may impose requirements on tail latency [8, 9]. Tail latency is defined to be the percentage of requests that can meet the desired latency requirement in a given time period. Also some applications can have higher priority level than others, where the priority can be defined in terms of their KPIs [8].

  The problem, then, is how to allocate resources to such applications in order to minimize operational monetary cost for MECs while meeting applications' KPIs [8, 9, 11]. In the literature, this resource allocation problem is also called an application placement problem [23] or a resource provisioning problem [24].

- **How to handle the heterogeneity of resource-constrained IoT devices in the MECs?**

  MECs will host applications from a wide range of resource-constrained devices, making it complex to migrate or port applications from resource-constrained devices to the cloud and vice-versa. There is a need for a framework that can provide an abstraction of these heterogeneous, resource-constrained devices in order to allow for seamless development, deployment, and management of applications [25]. The framework should provide an environment for homogeneous application development in such heterogeneous environments. The framework should also make it easy to offload part of an application either to the MECs or to the CDC to meet the application's requirements. The framework should also support simple development, deployment and management tools, to facilitate the life-cycle management of applications.

1.3 Research Methodology

The work described in this thesis has mainly been developed using the constructive research (CR) methodology [26, 27]. This methodology is used to solve domain-specific, practical problems while producing a theoretical contribution of academic value. The solutions – the "constructs" of the methodology – can be algorithms, models, theory, software or frameworks. The research process involves the following steps: (1) selecting a practically relevant problem; (2) obtaining a comprehensive understanding of the study area; (3) designing one or more applicable solutions to the problem; (4) demonstrating the solution's feasibility; (5) linking the results back to the theory and demonstrating their practical contribution; and (6) examining the generalizability of the results. CR essentially uses two forms of reasoning. Early stages of CR resemble deductive logic. For example, a single construct is designed from the vast amount of knowledge gained from a literature review of the study area. In the later stages, the reasoning follows inductive logic, when the results' theoretical and practical contributions, as well as their wider applicability, are considered.

We followed the general guidelines of the CR methodology. First, research problems of practical relevance for the resource allocation problem in MECs were identified, as described in Section 1.2. We performed a systematic literature review to understand what has already been done to solve similar or related problems in the domain. Our constructs take the form of algorithms, models and frameworks intended to minimize the cost for allocating resources to applications in MECs while meeting the applications' KPIs. The scientific contributions of our solutions are described in Section 1.4. We evaluated the proposed contributions via simulation and the creation of a software framework, in order to demonstrate the feasibility and practical relevance of our proposed solutions, and compared them with existing solutions. We investigated the generalizability of the proposed solutions by varying the parameters in the simulation environment.

1.4 Research Contribution

The goals of this thesis are: to study resource allocation problems for applications hosted by MECs, where the objective is to minimize system cost while satisfying the performance targets of applications; to propose models and algorithms for solving resource allocation problems; to develop simulation frameworks to implement and analyze the performance of our algorithms; and to develop a software framework to support the allocation of IoT applications running (in part) on resource-constrained devices. This thesis describes how we sought to realize these goals.

The first part describes models for MEC resources and application resource demands, spike detection methods, and workload generation models (Paper I, Paper II, Paper III, Paper V, and Paper VI). Paper II uses the models for spikes developed in Paper I in order to generate representative resource demands for applications and study the resource allocation problem in MECs. In the second part, we formulate the resource planning problem for cost-optimal deployment of applications (Paper III). The third part describes the optimization methods and distributed algorithms we used to solve the resource allocation problem for MEC (Paper II, Paper V and Paper VI). In part four, we explain how we extended the Calvin framework in order to provide support for resource-constrained devices, and how our framework can provide homogeneous application development, deployment and management in heterogeneous environments that include resource-constrained IoT devices, MECs and CDC (Paper IV). The contributions of this thesis are described in Section 4.

1.5 Thesis Outline

Section 2 describes opportunities and challenges related to modeling and resource allocation for applications in MECs. Section 3 describes a framework for the development, deployment and management of applications in MECs. Section 4 provides a summary of contributions.

2 Resource Modeling, Planning, and Allocation in Mobile Edge Clouds

MECs have yet to be deployed and MEC configurations yet to be defined. Thus, in order to study resource allocation problems in MECs, we must first develop a realistic model for MECs, incorporating compute resources, network topology and processing of application requests. Modeling of application resource demands and MEC infrastructure's resources is described in Section 2.1. Resource planning and allocation problems are described in Section 2.2.

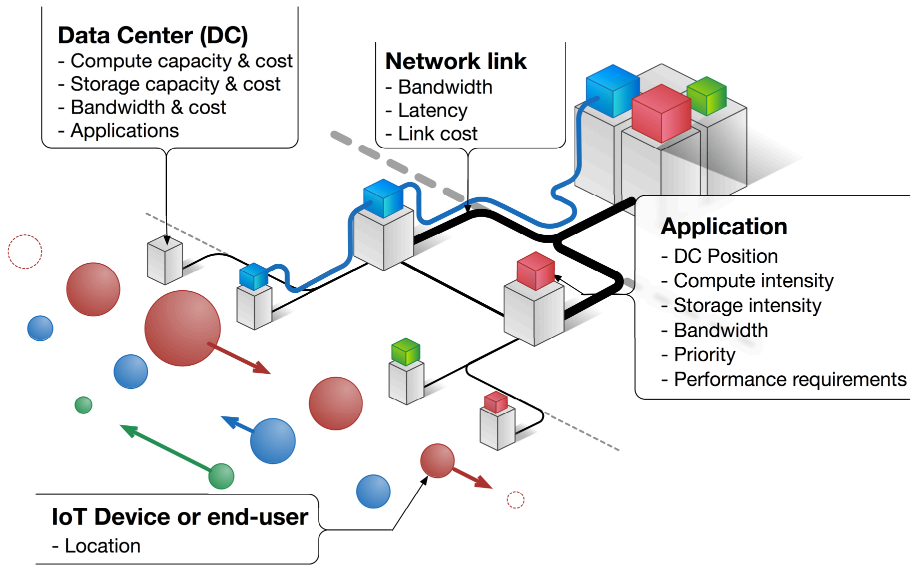

Figure 4: The entities and parameters relevant to resource allocation problems for MECs.

2.1 Application and Infrastructure Modeling

Modeling of application resource demands and infrastructure resources is essential to the study of resource allocation in MECs. The main entities and parameters relevant to resource allocation in MECs are shown in Figure 4.

2.1.1 Application Modeling

A wide range of applications will need to be served by MECs. Hence, it is vital, for resource allocation and planning, to understand the variations in applications' resource requirements with respect to time, space, and resource type.

An application's resource demands can be modeled in terms of two distinct aspects. The first aspect, *request quality*, is the amount of resources of different types, such as compute and bandwidth, required to serve an application's request. The second aspect, *request variation*, corresponds to the statistical distribution of the number of requests received per unit of time and, if relevant, per region of space.

An application's *request quality* can be modeled as the average resource requirement over time per request with respect to the consumed resource type. Requests from some applications may consume large amounts of compute resource compared to bandwidth and vice versa. Modeling request quality helps in understanding what type of resources and how much capacity will be needed for handling an application's requests in order to fulfill the application's KPIs at a given time.

In order to model an application's *request variation*, we consider how the number of requests varies with time and location. For example, the origin of resource requests can change over time, which will lead to different resource demands over both time and space [28].

Differences in request variation and request quality can arise simultaneously. For example, the distribution of users at a university campus may vary across the day and the access point locations [29, 30]. An application can have periodic or non-periodic variations in the resource demands over time. Request variation over time is very common in web applications: Wikipedia, for example, experiences periodic increases in workload on their servers [22]. Also, the resource requirements for some applications may be affected significantly by external events [31]. For example, a music festival or sports event draws huge numbers of people to a particular location, which could trigger higher resource requirements around that location for certain applications.

Applications' request variation modeling can help in understanding the resource requirement patterns for applications over time. Time series data for an application's request has three components: trend, periodicity and residuals [22]. The trend can be estimated by computing moving averages or differencing the time series data. The periodicity can be computed using auto-correlation, whereas the stationary residual can be estimated using auto-regressive models [22].

Short-lived and significant increases in demand for computing resources are known as *spikes* or *bursts*. Spikes or bursts may occur as a result of known events – online stores expect spikes on their servers during black Friday, for example – but the characteristics of those spikes, such as how quickly they will arise and how large they will be, may be unknown. Pattern forecasting methods along with statistical tests can be used for early detection of spikes [21]. Many measures have been proposed to characterize spikes or bursts, including steepness, magnitude, slope, duration, spatial locality, the index of dispersion, and the sample entropy [31, 32, 33].

2.1.2 Infrastructure Modeling

MECs are intended to provide access to high-performance compute, network, and storage resources as close as possible to end-users and IoT devices [1]. The expectation is that MECs will have a tree topology, as mobile core and access networks often take shape of fat trees [11, 12, 34]. The main aspects of a distributed MEC infrastructure to be modeled are compute, bandwidth, and storage capabilities of all DCs, including micro DCs at the edge, and the network characteristics of the links connecting them. A cost model for each entity (DC or link) can be expressed as a function of general capacity and capability properties, such as compute units, storage units, and bandwidth units [7, 12]. These cost models together with a general network topology can be used to simulate a distributed infrastructure for MECs. These system models together with the applications' resource demands and performance targets, can be used to study cost-optimal resource-allocation algorithms for applications in MECs. For infrastructure planning problems, we can use these models to determine the required capacity for MECs for cost-optimal allocation of applications.

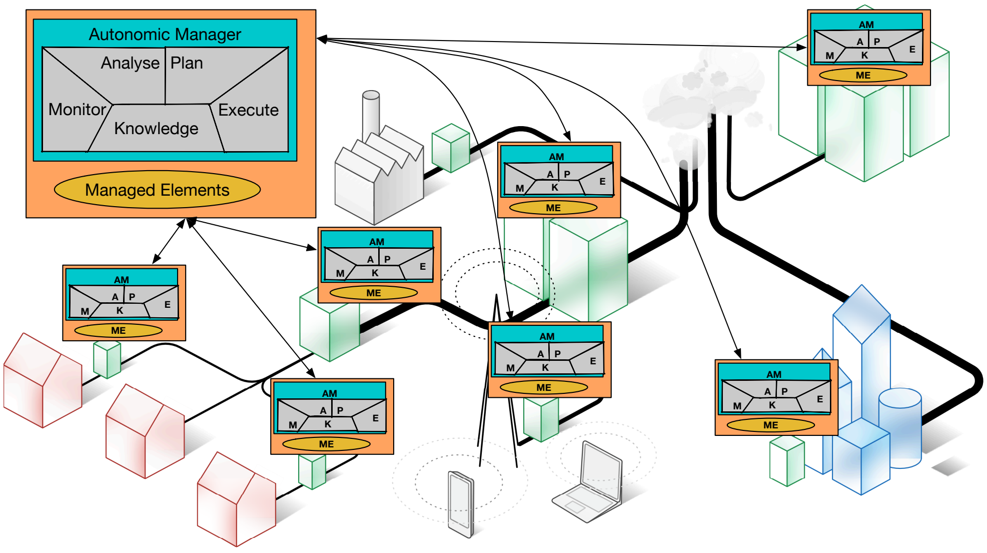

Figure 5: A hierarchical autonomic system for MECs.

2.2 Resource Planning and Allocation

The scale, complexity and resource-heterogeneity of an MEC means it is natural, from an architectural and management perspective, to view an MEC as a collection of self-adaptive or autonomic systems with multiple MAPE-K (Monitor, Analyze, Plan, Execute, and Knowledge) loops. This view is illustrated in Figure 5.

Autonomic systems comprise an interactive collection of autonomic elements [35]. For MECs, autonomic elements could be the geo-distributed compute resources, having heterogeneous compute capacity and cost, at the network edge. Each autonomic element may consist of one or more managed elements combined with a single autonomic manager that controls them. In this context, the managed elements are the hardware resources, whose characteristics, such as processing speed, bandwidth and storage, define the capabilities of an MEC. Each managed element is modified to enable the autonomic manager to monitor and control it. For the execution phase, actuators for the autonomic system may vary depending on the problems under study. For example, an MEC, viewed as an autonomic system, may have actuators for (1) auto-scaling (how much), and (2) migration (where), which contribute to solving the resource allocation problem. In general, the components of the MEC should collaborate to minimize the global cost of running applications inside the infrastructure.

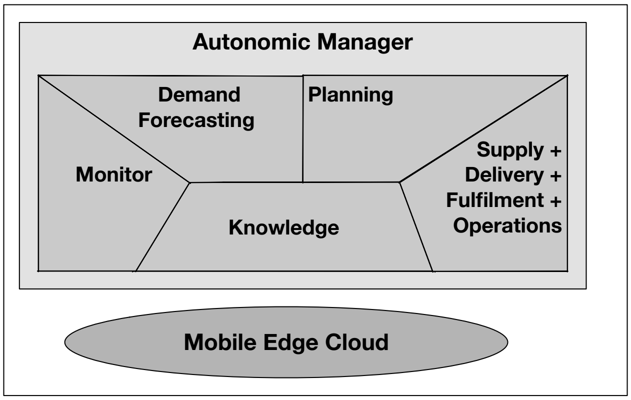

Figure 6: A variant of the MAPE-K loop for resource planning for MEC.

2.2.1 Resource Planning

Setting up the infrastructure for MECs requires huge investments[1], so it is important to decide *how much* capacity to add in terms of number of physical servers and *where* to locate them in order to efficiently manage applications. Constructing a DC campus may require planning of compute capacity, network fabric and WAN endpoints, cooling systems, power systems, and building management systems [3]. A variation of the MAPE-K loop for the capacity planning problem is shown in Figure 6. Here, the analysis phase is replaced with resource demand forecasting, and the execution phase is replaced with supply, delivery, and fulfillment, and operations of compute resources.

Forecasts of relevant information, based on network traffic and resource usage of existing applications, can be used when planning to augment resources of an existing DCs or add a new DCs to the MECs. Forecasts can be influenced by other factors, such as emerging applications, new features of existing applications, user growth, state of the software and hardware efficiency. For example, if there is huge increase in the number of emerging applications, then resource planning should include this in the forecasts for resource consumption.

2.2.2 Resource Allocation

Allocating an appropriate amount of resources at the right time and right place to each application, in such a way that the end-user's performance requirements are met, while minimizing the operational cost in MECs, and doing it in a scalable way, is a challenging problem. Resource allocation also has to be done quickly in order to accom-

[1]For example, it costs more than 1 billion dollars for Google to build a DC campus [36].

modate rapid variations in applications' workload. The heterogeneity of distributed resources inside MECs introduces diversified cost, capacity, and latency of network resources, storage cost, and cost, capacity, and speed (clock rate) of compute resources. The increased resource plurality also brings several resource management challenges for MECs [37]. The heterogeneous resources need to be autonomically managed with feedback from external and internal inputs for efficient resource utilization.

Algorithms to determine the allocation of resources to applications can be implemented either in a centralized, decentralized, or hierarchical manner [38, 39]. In centralized resource allocation, there is a single MAPE-K loop for MECs. All monitored data is collected at a centralized location and analyzed in order to derive the current cost of running the whole infrastructure, and determine the mapping of application components onto the MECs for cost-optimal operation. This approach may suffer from a single point of failure and may not scale well. In contrast, fully decentralized resource management enhances scalability, at the expense of a global overview of the system, which may lead to suboptimal resource allocation.

There exist different design patterns for designing MAPE-K loops in which the self-adaptive functions of the MAPE-K loops are decentralized [40], such as master-worker, coordinated control, and hierarchical control [38].

For a master-worker pattern, a single master component may run part of the self-adaptive functions, such as the analyze and plan phases, whereas multiple independent workers run the remaining functions, such as monitor and execution phases, in a decentralized manner. This pattern may suffer from bottlenecks due to the centralized components.

The coordinated control pattern consists of multiple control loops where each has a limited view of the system and loops must coordinate to achieve global, non-conflicting objectives. Depending on the degree of coordination, systems may take time to converge to a unified view and be slow to adapt to changes [38].

The hierarchical control pattern uses a layered architecture where MAPE-K loops in different layers perform different tasks, often on different time scales, to achieve different objectives. For example, the lower layers could work on shorter time scales and focus on the allocation of resources to applications, whereas higher layers can work on longer time scales and manage the whole system by scheduling system-wide reconfigurations. The hierarchical structuring of control nodes in a network topology and sharing of contextual information of the current system state, could reduce communication cost, while simultaneously maintaining a global overview of the system [41]. The hierarchical approach scales better than the centralized approach, and can also have full overview of the system [39], potentially making it suitable for resource allocation for MECs, given their scale and complexity [38].

The optimal resource allocation problem can be formulated in different ways in order to achieve various objectives [42]. For *Intra- and Inter-Data Center Placement* [43] in the context of Virtual Machine (VM) placement within DCs, the objectives could include minimization of latency [44], energy [45], network traffic [46], and communication cost [47]. These objectives could be realized through various strategies, such as load balancing [48], VM consolidation [49], data and end-user locality [50], and application affinity information [51]. The optimal resource allocation problem is an NP-hard [52, 44], so various heuristics to solve the problem have been proposed [44, 45, 49]. Some other factors that can influence resource allocation decisions include average and tail response time for applications [53], and the types of applications allocated to a compute resource [54].

Depending on the goals of the system, the objectives for resource allocation problems in MECs might include minimization of applications' RTT [55, 56] or system cost [15, 57, 58, 59, 60]. The objective could also be to maximize the edge resource utilization [61] or revenue for service providers [62]. The RTT minimization problem can be studied under various constraints, such as the (fixed) capacity of compute resources [55, 56]. System cost may include operational costs due to energy consumed during communication and processing phases [15, 57, 58], which may depend on many factors, such as end-users' mobility, network condition, utilization of compute and network resources, security and privacy requirements. Resource allocation problems can be studied subject to other constraints, such as applications' latency requirements [57] or both latency and capacity constraints for compute and network resources [58]. Depending on the applications' placement at the edge or the cloud, energy is consumed at different locations, such as cellular uplink, backhaul uplink, processing at the cloud node, backhaul downlink, and cellular downlink [58]. The system cost may also include reconfiguration and migration costs along with operational costs and applications' KPIs [60, 59]. Dynamic resource allocation is required over time to handle both applications' request quality and variations [15].

Content Delivery Networks (CDNs) share similar network topologies to MECs, but they focus on different types of applications. For CDNs, the content is static and the primary resources are storage and bandwidth. The bandwidth consumed when serving a particular item of content will typically be proportional to the number of times it is requested (that is, the demand for that content). The sizes of items of content hosted by a CDN might vary significantly, but the storage and CPU requirements are not, in any meaningful sense, proportional to the demand for content. The objective for a CDN could be the minimization of storage and bandwidth cost [63] for the content placement (using various caching techniques [64]) while meeting the latency requirements of content subscribers. In contrast, the resource management problem for MECs involves heterogeneous resources and applications with heterogeneous performance requirements, resulting in resource allocation problems being more complex in MECs.

In this thesis, we study the resource allocation problem in MECs, where the system goal is to minimize the overall monetary operational cost over time while meeting applications' performance targets [8, 9, 11]. To solve the resource allocation problem, performance and other metrics, such as operational cost, are monitored for both applications, and compute and network resources of MECs. One can use a "heartbeat" approach to gather the relevant information periodically for a centralized resource allocation algorithm for a MEC. For example, the heartbeat message can be sent periodically to carry the cost information about each entity (node or link) in the system. The system cost can be re-evaluated at every heartbeat to find the cost-optimal placement of applications. Lowering the heartbeat period could result in better application performance. However, it may incur higher bandwidth cost due to an increase in the number of application migrations, and performance penalties due to application downtime.

3 Framework for Life-cycle Management of IoT applications in Mobile Edge Clouds

The development of applications that can run seamlessly on IoT Devices and MECs could face challenges due to the heterogeneity of devices; multiple software platforms, communication protocols and programming languages for devices; and the complexity of distributed computing. For example, the development of current state-of-the-art IoT applications: (1) assumes that all processing performed by cloud and IoT devices is for sensing or actuating; (2) does not allow resource sharing between IoT devices; (3) assumes the deployment of applications is static; (4) are insufficiently abstracted, as developers need to specify IP-addresses, transport protocols, and device parameters; and (5) requires some programming effort to write a simple application [65].

A framework is required that can ease the problem of distributed application development, deployment, and management on geo-distributed computing resources ranging from small resource constrained devices to compute resources in MECs or CDCs. Such a framework could help in bridging the gap between different standards by providing (1) an abstraction layer for compute resources and platform features to applications, and (2) a common communications interface independent of physical methods [66]. It could also provide a *runtime* as and when it is needed to support heterogeneous devices or nodes in MECs or CDCs. Hence, it can improve efficiency and interoperability between different platforms and systems for running IoT applications in scenarios such as smart cities [67, 68]. The framework should also provide lightweight virtualization capabilities for IoT devices to support multi-tenancy, fast migration capabilities, and resource-constrained devices.

Such a framework should also support separation of concerns between the development of applications and the management of those applications on compute resources. One way to achieve this separation of concerns, is by using Actor and Flow-based programming models [69, 70]. A framework based on such models could also address the challenges of fragmentation in the development of IoT applications, mitigate the complexity of deployment and resource management of applications, and provide an abstraction layer to handle heterogeneity of compute resources.

There exist frameworks based on Actor and Flow-based programming models. For example, NoFlo [71] and Node-RED [72] use Flow-based programming to represent an application as a graph, in which nodes model processing components and edges model data flows between processing components within the application. Using Actor-based models, Orleans [73] provides a programming model, a runtime, and access to persistent, reliable, and scalable actors. Using Actor and Flow-based programming models, Calvin [65, 74, 75] provides a simplified application development environment for developers, a runtime, and requirement matching functionalities for the deployment of applications. Calvin also handles resource management for applications.

There exist frameworks, such as AWS Lambda [76], Apache OpenWhisk [77], and IFTTT [78], that are based on an asynchronous and event-driven architecture. These frameworks use three concepts: *triggers* (events that may occur); *actions* (what is done in response to the events); and *rules* (combinations of triggers and actions). They offer similar basic functionalities to Actor and Flow-based programming models in the sense that triggers and actions correspond to actors, and rules are equivalent to connections between the actors. In this thesis, we use the Calvin framework to study our research problems, as it is an open source project and also supports requirement-based deployment of applications on IoT devices in order to allow greater level of autonomic management.

The Calvin framework is described in Section 3.1. Then, some challenges related to Calvin framework that we address in this thesis are described in Section 3.3.

3.1 The Calvin Framework

Calvin is an open source peer-to-peer framework for the development, deployment, and execution of distributed IoT applications [65, 74, 75]. It also supports dynamic application deployment based on matching the requirements of applications with the capabilities of devices or nodes in the MECs or CDCs [65]. For example, in a smart city scenario, a video-surveillance application may state a requirement to be deployed on all traffic lights capable of capturing video in the city. The Calvin framework implementation is language-agnostic, as long as the implementation follows the inter-runtime communication protocols and accepts the common set of control commands [75]. Currently, the Calvin framework provides a runtime that is compatible with a wide range of hardware architectures, from high-performance nodes in the cloud to Raspberry Pi [65, 74, 75].

3.1.1 Application Development in Calvin

Calvin simplifies application development using the Actor and Flow-based programming models [69, 70]. Actor models encapsulate functionality and state in *actors*, and provide well defined interfaces, known as *ports*, to enable communication between actors [69]. Figure 7 illustrates an actor in the Calvin framework. An action is performed on tokens arriving at input ports and sent to output ports. It can also be triggered by an event, such as a timer, generated by a device where the actor is deployed on a Calvin runtime.

The Calvin framework divides application development and deployment into four phases – (1) *Describe*, (2) *Connect*, (3) *Deploy*, and (4) *Manage*. In the first phase, an application is described as a collection of functional units, *actors*, where each unit can provide data generation (e.g. sensors), processing or consumption (e.g. actuators). Actors can have internal states which are accessible to the actor itself. An actor can access the platform features through the Calvin runtime using the Hardware

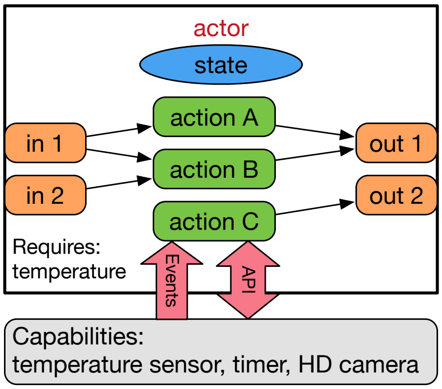

Figure 7: Inside an actor, an action is performed on tokens (or data) arriving at input ports (in 1 and in 2) or can be triggered by an event (e.g. timer).

Abstraction Layer (HAL). In the second phase, connections between the relevant actor ports are identified to represent data flow inside the application. The rules for requirement matching for dynamic deployment can also be specified during this phase [70, 79]. In the third phase, actors are allocated to distributed Calvin runtimes by matching actor requirements with the capabilities of the various Calvin runtimes. The fourth phase provides resource management for applications, such as auto-scaling and replication of actors on the available runtimes. The idea is to separate what tasks an application wants to perform, from where to deploy the application and how to perform the tasks. Moving the complexity from the application description makes sure that each application is independent of hardware dependencies and delegates resource management to the framework.

Using the Actor model and Flow-based programming, the Calvin framework also achieves isolation between data transport and data processing. The framework also enables portability between different platforms, as only the format of the data passed between the ports is standardized; how data is processed inside an actor is unimportant [75].

Hence, with the support of above mentioned features, the Calvin framework provides a clear separation of concerns among different entities, such as device manufacturers, application developers, marketplaces through which applications are distributed, and operators that deploy and maintain applications.

3.2 The Calvin Runtime

The Calvin runtime provides platform abstraction, an application execution environment, resource management, and data transport facilities. Calvin runtimes connect to each other, creating the illusion of a single, global runtime to application devel-

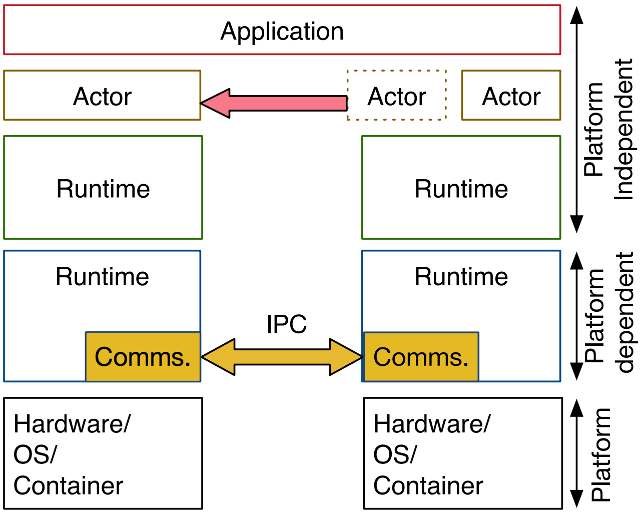

Figure 8: A Calvin runtime stack [75].

opers. Collectively, they take care of the complexity of application deployment and management on the distributed infrastructure.

The current version of the Calvin runtime is supported by most common operating systems (OSs), VMs, and containers. Figure 8 shows the high-level architecture for Calvin, part of which is platform-dependent and part platform-independent. The platform-dependent part provides data transport services and coordination between runtimes, as well as support for different kinds of transport layers, such as Bluetooth and WiFi, via a plug-in mechanism. It also provides an abstraction layer for hardware features, such as temperature sensors and HD cameras. The platform-independent part exposes platform resources to actors in a uniform manner and also provides resource management functionalities, such as actor replication and migration, and scheduling of actor's actions on their tokens.

An application, comprising actors and connections between actors, is deployed on a Calvin runtime and commences execution immediately if the actors' requirements are fulfilled. If the runtime does not meet the actors' requirements, then the actors are migrated automatically to runtimes where their requirements are met. By default, the Calvin framework uses a registry based on Distributed Hash Table (DHT) technology to maintain information about runtimes, actors, ports and any other data related to deployed applications.

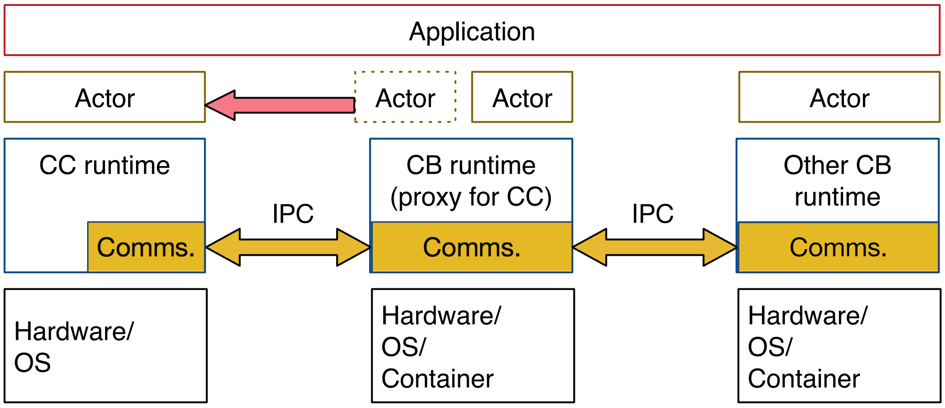

Figure 9: Calvin Constrained (CC) may rely on the Calvin-base (CB) for advanced features like actor placement decisions and access to application and resource registry.

3.3 Challenges for the Calvin Framework

The thesis addresses two challenges for the Calvin framework: support for resource-constrained devices and resource allocation for applications. We describe these challenges in more detail in the following sections.

3.3.1 Support for Resource-Constrained Devices

Ideally, the Calvin framework would support every type of device or node in an MEC or CDC, irrespective of its characteristics. The reference implementation for the Calvin framework, *Calvin-base*, is written in Python [74]. Calvin-base supports a wide range of platforms, from high-performance compute resources to IoT devices having CPUs with as little as 0.5 GB of memory and a clock speed of 700 MHz. However, many resource-constrained devices have extremely limited memory and processing power, typically based on a micro-controller with less than 100 kB of memory and a processor with a clock speed of 100 MHz. Thus an implementation of the Calvin framework with support for such devices is required.

The Calvin framework we propose for resource-constrained devices, *Calvin Constrained*, will only be able to support a limited subset of the features in Calvin-base. It will require a *Calvin-base* as a proxy to provide other, more advanced, functionalities, such as actor placement decisions, resource and application registry, and security. The interaction between Calvin-base and Calvin Constrained is illustrated in Figure 9.

3.3.2 Resource Allocation for Applications in a Calvin-type Framework

An application may have various requirements, both in terms of hardware support and performance, that a device on which the application is deployed should be able to meet. Examples of hardware requirements might include temperature sensors, GPUs and HD cameras. Performance requirements are often expressed as a set of KPIs for the application. The Calvin framework currently supports hardware, security, and locality requirements, but not performance requirements.

Thus there is a need to study algorithms for resource allocation in a Calvin-type framework that are able to minimize the operational cost for deploying applications while satisfying the applications' performance requirements. The problem of allocating resources to applications in such a framework can be formulated as an resource allocation problem where the objective is to minimize the monetary operational cost of all applications while meeting their performance targets [8, 9]. Moreover, distributed algorithms for resource allocation need to be investigated and developed in order to support efficient, large-scale deployment for such applications. It is expected that MECs will host applications with widely varying performance requirements. Algorithms based on service differentiation should also be investigated in order to handle this diversity [9].

4 Summary of Contributions

This thesis provides a comprehensive set of models to capture various aspects of MEC resources and applications. This thesis also provides a simulation framework to study the resource allocation and resource planning problems. This thesis covers various aspects of applications, such as scale, mobility, and heterogeneity in the resource-type and latency requirements. It also covers various aspects of the infrastructure, such as heterogeneity in the cost and performance of distributed resources.

MEC resource and infrastructure models capture the heterogeneity in capacity as well as the cost of MECs' resources. The cost of heterogeneous resources in a MEC is also compared with the charges made by major cloud providers for similar resources in order to validate the cost models. The application models capture workload quality, workload variation and performance requirements for IoT applications. We have performed sensitivity analyses for parameters in the application models to understand their impact on cost-optimal resource allocation algorithms. We also provide a linear optimization formulation to study the resource planning problem for cost-optimal operations for MECs.

Paper 1: Online Spike Detection in Cloud Workloads
In this work [21], we investigate online workload spike detection methods using adaptive signal processing techniques, such as auto-regression, double exponential smoothing, low-pass filtering, and the use of constant mean values. We derive a number of workload models coupled with a stopping rule that signals whenever there is change in

the system model. The stopping rule is based on a cumulative sum (CUSUM) test [80] for detecting the onset of a spike. The main idea is to detect the spike as early as possible in order to maximize the time available for mitigating its effects. We explore the trade-off between the ability to correctly detect large traffic increases as spikes and how prone the detectors are to generate false alarms when spikes are not present. We also explore the trade-off between how early detection can be made, and the number of false alarms generated.

We evaluate the online detectors by calculating *precision* and *recall* metrics using the number of hits and misses. Additionally, we measure the Average Time Before Peak (ATBP) in order to evaluate the speed with which detectors can detect a true spike. We also measure the Average Relative Change (ARC), which is the percentage relative change between the peak workload level and the workload level at the time of detection for a true spike. Some methods detect spikes early but generate large numbers of false alarms, whereas others are slower to detect spikes but generate fewer false alarms. These trade-offs need to be carefully considered by an application owner or service provider: for example, methods that detect spikes very early while generating large number of false alarms can lead to low resource utilization during false alarms but better performance of applications during spikes.

I was the main author of the paper. Regarding my own contributions to this paper, I proposed the framework for spike detection by combining adaptive signal processing techniques, such as auto-regression, double exponential smoothing, and low-pass filtering, with a statistical test method (CUSUM). I received feedback from co-authors of the paper during the problem formulation, writing, and evaluations.

Paper 2: Dynamic Application Placement in the Mobile Cloud Network

In this paper, MEC refers to a *mobile cloud network*. Cloud providers and Telecom-operators foresee MECs as a way of addressing the challenges of consistent performance and low communication latency requirements for IoT applications as well as a high degree of user mobility, while minimizing their operational cost. In this work [37], we formulate a set of MEC management problems and address the challenges arising from high user mobility, cost and capacity heterogeneity of compute resources, and a highly distributed infrastructure.

The paper makes three significant contributions. Firstly, the paper contributes models that capture the cost- and capacity-heterogeneity of an infrastructure that incorporates MECs by proposing multiple tiers of resource types, such as compute resources at base stations and DCs. A set of resource management challenges is also identified based on the infrastructure models. Secondly, a cost-based optimization formulation for resource allocation is presented, in which computation and communication costs are minimized, assuming that communication latency is a hard constraint. Thirdly, a local search-space based algorithm is proposed in order to address the lack of scalability of centralized approaches.

The placement algorithms are evaluated based on metrics such as monetary cost for deploying applications, average RTT for applications, and average resource utilization of the compute and network resources. We simulate three scenarios: *Mobile users*, *Daily commute pattern for a university campus*, and *Large one-time arena event*. The evaluation demonstrates that the proposed iterative local-search based placement algorithm significantly reduces latency and improves resource utilization skewness while minimizing the operational cost of the system. For example, in the *Mobile users* scenario, our (dynamic) allocation algorithm, based on iterative, local search, provides up to 25% reduction in cost, on average, compared with a static allocation strategy based on an initially-optimal placement of applications.

I was one of two equal main authors of the paper. The problem formulation was jointly derived with co-authors. I contributed to the development of system models, workload models, optimization method, and the dynamic application placement algorithm based on local search. I contributed equally to the development of the simulation framework. I also received feedback from co-authors of the paper during writing and evaluations on my part.

Paper 3: How Beneficial are Intermediate Layer Data Centers in Mobile Edge Networks?

In this paper, MEC refers to a *mobile edge network*. As capital and operational expenses for a DC are huge, it is important to understand the benefits of adding additional DCs in a MEC. In this paper [7], we study where additional DCs should be deployed in the MEC infrastructure in the interests of reducing total costs. We also identify relevant system parameters in a MEC and investigate the sensitivity of these parameters, based on average resource consumption, application demand distribution, and resource cost distribution. The investigation shows that the most important parameter when determining whether intermediate layer DCs are beneficial is application type, with demand locality and variation in the DC's resource cost across the MEC topology being the second and third most sensitive, respectively.

The investigation also shows that having an additional layer of DCs at the edge could save up to 67% of the cost for bandwidth-hungry applications in the evaluated scenarios. Hence, it may be beneficial for a Telecom operator to build an edge DC having up to a hundred nodes at the network edge to cater for bandwidth-hungry applications and to minimize its operational cost.

I was the main author of the paper. In this paper, I contributed to the problem formulation, system dynamics including cost and sensitivity analysis, and evaluations for the resource planning problem. I received feedback from co-authors of the paper during the problem formulation, writing, and evaluations.

Paper 4: Calvin Constrained – A Framework for IoT Applications in Heterogeneous Environments

Calvin [74] is a peer-to-peer framework for application development, deployment and management in heterogeneous environments, including IoT devices and high-performance nodes in MECs and CDCs. When using the Calvin framework for application development, developers do not need to worry about *how* something should be computed, instead the focus can be on *what* should be computed. The framework handles the complexity of deployment and management of resources and provides an abstraction layer for platform features to the applications.

In this work [25], we propose *Calvin Constrained*, an extension of the Calvin framework suitable for resource-constrained devices. Calvin Constrained only supports a limited but essential set of features of the reference framework, *Calvin-base*, such as actor deployment, migration, execution, data communication, and runtime resource abstraction. Calvin Constrained needs to be supported by the full *Calvin-base* runtime in order to provide services such as actor placement decisions, resource and application registry, and security infrastructure. Calvin Constrained supports actors implemented in C and Python, where support for Python actors is enabled by using *MicroPython* [81] as a statically allocated library. Support for Python actors on the runtime enhances code re-usability and enables the automatic management of state variables, but comes at the cost of increased runtime resource usage compared to actors implemented in C. We show that the extra resources needed when using the Calvin Constrained framework are manageable on current, off-the-shelf, microcontroller-equipped devices.

I was the main author of the paper. In this work, I analyzed a subset of features that the Calvin framework must support for resource-constrained devices. I contributed to the implementation supporting Python actors' deployment, migration and execution using MicroPython as a statically linked library in C. I evaluated the performance of the Calvin Constrained framework and the costs of supporting Python actors. I received feedback from co-authors of the paper during the problem formulation, writing, and evaluations.

Paper 5: Distributed Cost-Optimized Placement for Latency-Critical Applications in Heterogeneous Environments

MECs in 5G networks will transform many industrial sectors, such as transportation, healthcare, and manufacturing. The main factors fueling the growth in edge computing are the unprecedented growth of data, the rapid rise in the number of latency-critical applications, the introduction of more stringent regulations concerning the processing and management of personal data, and the emergence of a distributed computing architectures making use of specialized hardware such as GPUs.

In this work [8], we study how one can deploy and manage large numbers of latency-critical applications in MECs cost-efficiently, given the heterogeneity of devices, application performance requirements, and workloads. We also study cost and performance dynamics, and propose distributed algorithms for the automatic deployment of such applications in heterogeneous environments that may include IoT devices, MECs, and CDC. We evaluate placement algorithms with respect to a number of metrics, including number of required runtimes, application slowdown, and the number of iterations required to place applications in order to minimize operational cost while meeting the performance requirements of applications. Our results show that iterative search- and size-based distributed algorithms outperform random and bin-packing algorithms.

I was the main author of the paper. In the paper, I developed the simulation framework used to conduct the cost and performance sensitivity analysis. I also proposed distributed algorithms for the allocation of applications in MECs and CDC. I received feedback from co-authors of the paper during the problem formulation, writing, and evaluations.

Paper 6: Utility-based Allocation of Industrial IoT Applications in Mobile Edge Clouds

It is expected that MECs will run applications with a wide range of latency requirements. Some applications, such as autonomous vehicles, may require very low RTT, whereas video encoding applications may have less demanding requirements for RTT.

In this work [9], we propose a two-tier scheduling framework for allocating resources to IIoT applications. The higher-level scheduler runs periodically and decides – based on system state and the performance of applications – whether to admit new applications and/or migrate existing ones. In contrast, the lower-level scheduler decides which application will be assigned to the runtime resource next. We define performance-based metrics that determine to what extent the Service Level Objectives (SLOs) of an application are being met. The *Application Happiness* metric is based on a single application's performance and SLOs, and the performance of the network. The *Runtime Happiness* metric is derived periodically from the Application Happiness of all deployed applications the runtime is hosting. These happiness metrics enable a resource allocation algorithm to make decisions based on the extent to which the infrastructure is meeting the performance targets of the applications. These metrics may be used for decision-making by the scheduler, rather than runtime utilization, for example.

We evaluate four scheduling policies for the high-level scheduler in combination with five for the low-level scheduler, where the combined, two-tier scheduler's objective was to allocate resources to applications such that system cost is minimized while ensuring that applications' performance requirements are satisfied. The policies are evaluated with respect to the number of runtimes, the impact on the performance of applications, and utilization of the runtimes. For the evaluated scenarios, policies based on Runtime and Application Happiness outperform other policies for the schedulers, including the bin-packing and random strategies.

I was the main author of the paper. I developed the two-level scheduler, extended the simulation framework from Paper 4, and compared the results obtained using the new scheduler and happiness-based policies with those obtained in the earlier paper. I received feedback from co-authors of the paper during the problem formulation, writing, and evaluations.

References

[1] Jim Davis, Philbert Shih, and Alex Marcham. An Edge Computing Ecosystem Report. `https://www.stateoftheedge.com/`. Accessed: 2018-08-07.

[2] Mahadev Satyanarayanan. The Emergence of Edge Computing. *Computer*, 50 (1):30–39, Jan 2017.

[3] Luiz André Barroso, Jimmy Clidaras, and Urs Hölzle. The datacenter as a computer: An introduction to the design of warehouse-scale machines. *Synthesis lectures on computer architecture*, 8(3), 2013.

[4] AWS Regions and Endpoints - Amazon Web Services. `https://docs.aws.amazon.com/general/latest/gr/rande.html`. Accessed: 2018-08-07.

[5] Michael Geist. Canada's Usage Based Billing Controversy: How to address the Wholesale and Retail Issues. `http://www.michaelgeist.ca/wp-content/uploads/2011/03/GeistonUBB.pdf`. Accessed: 2016-05-09.

[6] Sean Kenneth Barker and Prashant Shenoy. Empirical evaluation of latency-sensitive application performance in the cloud. In *Proceedings of the first annual ACM SIGMM conference on Multimedia systems*, pages 35–46. ACM, 2010.

[7] Amardeep Mehta, William Tärneberg, Cristian Klein, Johan Tordsson, Maria Kihl, and Erik Elmroth. How Beneficial Are Intermediate Layer Data Centers in Mobile Edge Networks? In *2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, pages 222–229, September 2016.

[8] Amardeep Mehta and Erik Elmroth. Distributed Cost-Optimized Placement for Latency-Critical Applications in Heterogeneous Environments. In *Proceedings of the 15th IEEE International Conference on Autonomic Computing (ICAC)*, 2018.

[9] Amardeep Mehta, Ewnetu B. Lakew, Johan Tordsson, and Erik Elmroth. Utility-based Allocation of Industrial IoT Applications in Mobile Edge Clouds, 2018. *Submitted for publication.*

[10] Hang Liu, Fahima Eldarrat, Hanen Alqahtani, Alex Reznik, Xavier de Foy, and Yanyong Zhang. Mobile edge cloud system: Architectures, challenges, and approaches. *IEEE Systems Journal*, (99):1–14, 2017.

[11] William Tärneberg, Amardeep Mehta, Eddie Wadbro, Johan Tordsson, Johan Eker, Maria Kihl, and Erik Elmroth. Dynamic Application Placement in the Mobile Cloud Network. *Future Generation Computer Systems*, 70:163–177, May 2017.

[12] William Tärneberg, Alessandro Vittorio Papadopoulos, Amardeep Mehta, Johan Tordsson, and Maria Kihl. Distributed approach to the holistic resource management of a mobile cloud network. In *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, pages 51–60. IEEE, 2017.

[13] Michael T. Beck, Martin Werner, Sebastian Feld, and Thomas Schimper. Mobile Edge Computing: A Taxonomy. In *Proc. of the Sixth International Conference on Advances in Future Internet*, 2014.

[14] Shiqiang Wang, Murtaza Zafer, and Kin K Leung. Online placement of multi-component applications in edge computing environments. *IEEE Access*, 5:2514–2533, 2017.

[15] Shiqiang Wang, Rahul Urgaonkar, Ting He, Kevin Chan, Murtaza Zafer, and Kin K Leung. Dynamic service placement for mobile micro-clouds with predicted future costs. *IEEE Transactions on Parallel and Distributed Systems*, 28(4):1002–1016, 2017.

[16] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, pages 13–16. ACM, 2012.

[17] Luis M Vaquero and Luis Rodero-Merino. Finding your way in the fog: Towards a comprehensive definition of fog computing. *ACM SIGCOMM Computer Communication Review*, 44(5):27–32, 2014.

[18] Enrique Saurez, Kirak Hong, Dave Lillethun, Umakishore Ramachandran, and Beate Ottenwälder. Incremental deployment and migration of geo-distributed situation awareness applications in the fog. In *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems*, pages 258–269. ACM, 2016.

[19] Hoang T Dinh, Chonho Lee, Dusit Niyato, and Ping Wang. A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless Communications and Mobile Computing*, 13(18):1587–1611, 2013.

[20] Peter Bosch, Alessandro Duminuco, Fabio Pianese, and Thomas L. Wood. Telco clouds and Virtual Telco: Consolidation, convergence, and beyond. In *2011 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 982–988, May 2011.

[21] Amardeep Mehta, Jonas Dürango, Johan Tordsson, and Erik Elmroth. Online Spike Detection in Cloud Workloads. In *2015 IEEE International Conference on Cloud Engineering (IC2E)*, pages 446–451, March 2015.

[22] Ahmed Ali-Eldin, Ali Rezaie, Amardeep Mehta, Stanislav Razroev, Sara S. de Luna, Oleg Seleznjev, Johan Tordsson, and Erik Elmroth. How will your

workload look like in 6 years? Analyzing Wikimedia's workload. In *2014 IEEE International Conference on Cloud Engineering*, pages 349–354, March 2014.

[23] Akshat Verma, Puneet Ahuja, and Anindya Neogi. pMapper: Power and Migration Cost Aware Application Placement in Virtualized Systems. In *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, Middleware '08, pages 243–264, New York, NY, USA, 2008. Springer-Verlag New York, Inc.

[24] Sivadon Chaisiri, Bu-Sung Lee, and Dusit Niyato. Optimization of Resource Provisioning Cost in Cloud Computing. *IEEE Transactions on Services Computing*, 5(2):164–177, April 2012.

[25] Amardeep Mehta, Rami Baddour, Fredrik Svensson, Harald Gustafsson, and Erik Elmroth. Calvin Constrained – A framework for IoT applications in heterogeneous environments. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 1063–1073. IEEE, 2017.

[26] Liisa Lehtiranta, Juha-Matti Junnonen, Sami Kärnä, and Laura Pekuri. The Constructive Research Approach: Problem Solving for Complex Projects. `http://www.gpmfirst.com/books/designs-methods-and-practices-research-project-management/constructive-research-approach`, Aug 2018.

[27] Gordana Dodig Crnkovic. Constructive research and info-computational knowledge generation. In *Model-Based Reasoning in Science and Technology*, pages 359–380. Springer, 2010.

[28] Tracy Camp, Jeff Boleng, and Vanessa Davies. A survey of mobility models for ad hoc network research. *Wireless Communications and Mobile Computing*, 2 (5):483–502, August 2002.

[29] David Kotz and Kobby Essien. Analysis of a campus-wide wireless network. *Wirel. Netw.*, 11(1-2):115–133, January 2005.

[30] Diane Tang and Mary Baker. Analysis of a Local-Area Wireless Network. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, MobiCom '00, pages 1–10, New York, NY, USA, 2000. ACM.

[31] Peter Bodik, Armando Fox, Michael J. Franklin, Michael I. Jordan, and David A. Patterson. Characterizing, Modeling, and Generating Workload Spikes for Stateful Services. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, pages 241–252, New York, NY, USA, 2010. ACM.

[32] Ningfang Mi, Giuliano Casale, Ludmila Cherkasova, and Evgenia Smirni. Burstiness in multi-tier applications: Symptoms, causes, and new models. In *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 265–286. Springer, 2008.

[33] Ahmed Ali-Eldin, Oleg Seleznjev, Sara S. de Luna, Johan Tordsson, and Erik Elmroth. Measuring cloud workload burstiness. In *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, pages 566–572, December 2014.

[34] Paul Bedell. *Cellular Networks: Design and Operation: A Real World Perspective*. Outskirts Press, 2014.

[35] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[36] Google Data Centers – Economic Impact and Community Benefit. `https://www.oxfordeconomics.com/recent-releases/d8d830e4-6327-460e-95a5-c695a32916d9`. Accessed: 2018-08-07.

[37] William Tärneberg, Amardeep Mehta, Johan Tordsson, Maria Kihl, and Erik Elmroth. Resource Management Challenges for the Infinite Cloud. In *10th International Workshop on Feedback Computing at CPSWeek*, 2015.

[38] Valeria Cardellini, Francesco Lo Presti, Matteo Nardelli, and Gabriele Russo Russo. Decentralized self-adaptation for elastic Data Stream Processing. *Future Generation Computer Systems*, 87:171 – 185, 2018.

[39] Hendrik Moens, Jeroen Famaey, Steven Latre, Bart Dhoedt, and Filip De Turck. Design and Evaluation of a Hierarchical Application Placement Algorithm in Large Scale Clouds. In *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops*, pages 137–144. IEEE, 2011.

[40] Danny Weyns, Bradley Schmerl, Vincenzo Grassi, Sam Malek, Raffaela Mirandola, Christian Prehofer, Jochen Wuttke, Jesper Andersson, Holger Giese, and Karl M. Göschka. *On Patterns for Decentralized Control in Self-Adaptive Systems*, pages 76–107. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[41] Jeroen Famaey, Steven Latré, John Strassner, and Filip De Turck. A hierarchical approach to autonomic network management. In *2010 IEEE/IFIP Network Operations and Management Symposium Workshops*, pages 225–232, 2010.

[42] Zoltán Ádám Mann. Allocation of virtual machines in cloud data centers—A survey of problem models and optimization algorithms. *ACM Computing Surveys*, 48(1):11:1–11:34, August 2015.

[43] Benny Rochwerger, Alex Galis, Eliezer Levy, Juan A Caceres, David Breitgand, Yaron Wolfsthal, Ignacio Martin Llorente, Mark Wusthoff, Rubén S Montero, and Erik Elmroth. RESERVOIR: Management technologies and requirements for next generation Service Oriented Infrastructures. In *2009 IFIP/IEEE International Symposium on Integrated Network Management*, pages 307–310, June 2009.

[44] Mansoor Alicherry and T. V. Lakshman. Network Aware Resource Allocation in Distributed Clouds. In *2012 Proceedings IEEE INFOCOM*, pages 963–971. IEEE, 2012.

[45] Anton Beloglazov, Jemal Abawajy, and Rajkumar Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing. *Future Generation Computer Systems*, 28(5):755–768, May 2012.

[46] Luiz F. Bittencourt, Edmundo R. M. Madeira, and Nelson L. S. Da Fonseca. Scheduling in hybrid clouds. *IEEE Communications Magazine*, 50(9):42–47, September 2012.

[47] Zheng Zhang, Ming Zhang, Albert G. Greenberg, Y. Charlie Hu, Ratul Mahajan, and Blaine Christian. Optimizing cost and performance in online service provider networks. In *7th USENIX Symposium on Networked Systems Design and Implementation*, pages 33–48, 2010.

[48] Martin Randles, David Lamb, and A. Taleb-Bendiab. A comparative study into distributed load balancing algorithms for cloud computing. In *2010 IEEE 24th International Conference on Advanced Information Networking and Applications Workshops*, pages 551–556, April 2010.

[49] Anton Beloglazov and Rajkumar Buyya. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in Cloud data centers. *Concurrency and Computation: Practice and Experience*, 24(13):1397–1420, September 2012.

[50] Sharad Agarwal, John Dunagan, Navendu Jain, Stefan Saroiu, Alec Wolman, and Harbinder Bhogan. Volley: Automated data placement for geo-distributed cloud services. In *7th USENIX Symposium on Networked Systems Design and Implementation*, volume 10, 2010.

[51] Lars Larsson, Daniel Henriksson, and Erik Elmroth. Scheduling and monitoring of internally structured services in cloud federations. In *2011 IEEE Symposium on Computers and Communications (ISCC)*, pages 173–178. IEEE, 2011.

[52] Sartaj Sahni and Teofilo Gonzalez. P-Complete approximation problems. *Journal of the ACM (JACM)*, 23(3):555–565, July 1976.

[53] Ewnetu Lakew, Cristian Klein, Francisco Hernandez-Rodriguez, and Erik Elmroth. Tail Response Time modeling and control for Interactive Cloud Services. 2015.

[54] Mina Sedaghat, Francisco Hernández-Rodriguez, Erik Elmroth, and Sarunas Girdzijauskas. Divide the task, multiply the outcome: Cooperative VM consolidation. In *Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on*, pages 300–305. IEEE, 2014.

[55] L. Tong, Y. Li, and W. Gao. A hierarchical edge cloud architecture for mobile computing. In *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9, April 2016.

[56] H. Tan, Z. Han, X. Li, and F. C. M. Lau. Online job dispatching and scheduling in edge-clouds. In *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, pages 1–9, May 2017.

[57] K. Zhang, Y. Mao, S. Leng, Q. Zhao, L. Li, X. Peng, L. Pan, S. Maharjan, and Y. Zhang. Energy-Efficient Offloading for Mobile Edge Computing in 5G Heterogeneous Networks. *IEEE Access*, 4:5896–5907, 2016.

[58] Ali Al-Shuwaili, Osvaldo Simeone, Alireza Bagheri, and Gesualdo Scutari. Joint uplink/downlink optimization for backhaul-limited mobile cloud computing with user scheduling. *IEEE Transactions on Signal and Information Processing over Networks*, 3(4):787–802, 2017.

[59] M. Chen, B. Liang, and M. Dong. Joint offloading and resource allocation for computation and communication in mobile cloud with computing access point. In *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, pages 1–9, May 2017.

[60] L. Wang, L. Jiao, J. Li, and M. Mühlhäuser. Online Resource Allocation for Arbitrary User Mobility in Distributed Edge Clouds. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 1281–1290, June 2017.

[61] Olena Skarlat, Matteo Nardelli, Stefan Schulte, and Schahram Dustdar. Towards QoS-aware fog service placement. In *Fog and Edge Computing (ICFEC), 2017 IEEE 1st International Conference on*, pages 89–96. IEEE, 2017.

[62] S. Guo, D. Wu, H. Zhang, and D. Yuan. Resource Modeling and Scheduling for Mobile Edge Computing: A Service Provider's Perspective. *IEEE Access*, 6: 35611–35623, 2018.

[63] Frank Schaffa and Jean-Paul Nussbaumer. On bandwidth and storage tradeoffs in multimedia distribution networks. In *IEEE INFOCOM '95. Fourteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Bringing Information to People. Proceedings*, pages 1020–1026 vol.3, April 1995.

[64] Spiridon Bakiras and Thanasis Loukopoulos. Combining replica placement and caching techniques in content distribution networks. *Computer Communications*, 28(9):1062–1073, June 2005.

[65] Ola Angelsmark and Per Persson. Requirement-based deployment of applications in Calvin. In *International Workshop on Interoperability and Open-Source Solutions*, pages 72–87. Springer, 2016.

[66] Soma Bandyopadhyay, Munmun Sengupta, Souvik Maiti, and Subhajit Dutta. Role of middleware for internet of things: A study. *International Journal of Computer Science and Engineering Survey*, 2(3):94–105, 2011.

[67] Sylvain Kubler, Jérémy Robert, Ahmed Hefnawy, Kary Främling, Chantal Cherifi, and Abdelaziz Bouras. Open IoT ecosystem for sporting event management. *IEEE Access*, 5:7064–7079, 2017.

[68] Jérémy Robert, Sylvain Kubler, Niklas Kolbe, Alessandro Cerioni, Emmanuel Gastaud, and Kary Främling. Open IoT Ecosystem for Enhanced Interoperability in Smart Cities – Example of Métropole De Lyon. *Sensors*, 17(12):2849, 2017.

[69] Carl Hewitt, Peter Bishop, and Richard Steiger. A universal modular ACTOR formalism for artificial intelligence. In *Proceedings of the 3rd international joint conference on Artificial intelligence*, pages 235–245. Morgan Kaufmann Publishers Inc., 1973.

[70] Johan Eker and J Janneck. CAL language report: Specification of the CAL actor language, 2003.

[71] NoFlo. `https://noflojs.org/`. Accessed: 2018-08-07.

[72] Node-RED. `https://nodered.org/`. Accessed: 2018-08-07.

[73] Sergey Bykov, Alan Geller, Gabriel Kliot, James Larus, Ravi Pandya, and Jorgen Thelin. Orleans: A framework for cloud computing. *Technical Report MSR-TR-2010-159*, 2010.

[74] Calvin. `https://github.com/EricssonResearch/calvin-base/`. Accessed: 2018-08-07.

[75] Per Persson and Ola Angelsmark. Calvin – Merging Cloud and IoT. *Procedia Computer Science*, 52:210–217, 2015. ISSN 18770509. doi: 10.1016/j.procs.2015.05.059.

[76] AWS Lambda – Run code without thinking about servers. Pay only for the compute time you consume. `https://aws.amazon.com/lambda/`. Accessed: 2018-08-20.

[77] Apache OpenWhisk – Open source serverless cloud platform. Executes functions in response to events at any scale.

[78] IFTTT – Build new services. `https://ifttt.com/`. Accessed: 2018-08-20.

[79] J. Paul Morrison. *Flow-Based Programming: A New Approach to Application Development.* CreateSpace, 2010.

[80] Fredrik Gustafsson. *Adaptive Filtering and Change Detection*, volume 1. Wiley, 2000.

[81] MicroPython. `https://micropython.org/`. Accessed: 2018-08-07.