UMEÅ UNIVERSITET

MANAGING CLOUD RESOURCE SCARCITY

Lars Larsson

Doctoral Thesis, October 2020
Department of Computing Science
Umeå University
Sweden

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

`https://people.cs.umu.se/~larsson/`
larsson@cs.umu.se

ISBN (print): 978-91-7855-328-0
ISBN (digital): 978-91-7855-329-7
ISSN: ISSN-0348-0542
UMINF: UMINF 20.07

The papers in this thesis have been re-typeset to match the overall style of the thesis with permission granted by the copyright holders.
Printed in Umeå, 2020.

“Hur det än blir så ska gudarna veta att det finns inga bojor eller band som kan hindra mina tankar att färdas norrut på våren, som en flygande and”

— Euskefeurat,
“Det är hit man kommer när man kommer hem”

Abstract

According to the Infrastructure-as-a-Service conceptualization of cloud computing, Infrastructure Providers offer utility-like pay-as-you-go access to computing resources (e.g., data processing, networks, and storage) to Service Providers, who use those resources to host applications for the benefit of end users. The quantity of resources available to Infrastructure Providers at any given moment is limited, as is the quantity of resources allocated to the applications of each Service Provider.

This thesis examines the management of cloud resource scarcity from the perspectives of both Infrastructure and Service Providers, with the aim of finding ways to ensure that the end user experience is minimally affected.

We consider three main strategies for managing cloud resource scarcity. First, we explore ways to efficiently construct collaborative *federations* of autonomous and independent Infrastructure Providers that allow local resource scarcity to be masked by extension using capacity from remote sites. Second, we consider how *scheduling* both within a cloud site and across a federation can be made aware of restrictions imposed by Service Providers for, e.g., performance or legal reasons. Third, we suggest ways of making applications conscious of resource availability so that they can apply *quality elasticity* under resource constraints.

The thesis is the culmination of 11 years of work within academia and industry. Based on the unique perspective granted by this long experience, the introductory chapters present a historical view of each subtopic mentioned above. Specifically, they discuss how cloud computing has evolved in conjunction with ways of developing applications to the symbiotic benefit of both, leading to the emergence of *cloud-native* software that allows Infrastructure Providers to use their infrastructure more efficiently and offer it more affordably while simultaneously granting Service Providers improved availability and performance in cloud-based environments.

Sammanfattning

I molnet (cloud computing), ur Infrastructure-as-a-Service-perspektivet, ger infrastruktursleverantörer tillgång till datorresurser (såsom databehandlings-, nätverks- och lagringskapacitet) med en löpande betalningsmodell till tjänsteleverantörer så att dessa kan erbjuda applikationer till fördel för slutanvändare. För infrastruktursleverantörer finns det en begränsning i mängden datorresurser som är tillgängliga vid varje givet tillfälle. För tjänsteleverantörer finns det en begränsning i hur många sådana resurser som är allokerade till deras applikationer.

Det övergripande målet med denna doktorsavhandling är att studera olika sätt att hantera resursbrister i molnet ur både infrastruktursleverantörers och tjänsteleverantörers perspektiv, så att slutanvändares upplevelse blir minimalt påverkad.

Våra ansatser till att hantera resursbrister i molnet kommer från tre huvudsakliga områden. Först utforskar vi *federationer* av autonoma och oberoende infrastruktursleverantörer, som möjliggör att lokala resursbrister kan maskeras genom att utöka och nyttja resurser från andra leverantörer. Därnäst undersöker vi hur *schemaläggning* både inom och mellan molnsajter i en federation kan göras medveten om begränsningar som tjänsteleverantörer kräver av exempelvis prestanda- eller legala skäl. Slutligen föreslår vi olika sätt hur applikationer kan göras medvetna om nuvarande resurstillgång och göras *kvalitetselastiska*.

Avhandlingen utgör kulmen av 11 års arbete inom akademins och industrins värld. Baserat på de unika möjligheter en sådan erfarenhet ger presenteras även ett historiskt perspektiv av dessa områden i de inledande kapitlen. I dessa kapitel diskuterar vi hur molnet har utvecklats tillsammans med hur applikationer levereras till slutkunder och hur ett symbiotiskt förhållande uppstått dem emellan. Resultatet är mjukvara som är *cloud-native*, vilket möjliggör för infrastruktursleverantörer att till högre grad effektivt utnyttja sin infrastruktur och erbjuda tillgång till den på ett mer kostnadseffektivt sätt, samt ger tjänsteleverantörer ökad tillgänglighet och prestanda i molnbaserade miljöer.

Preface

The research goal of this thesis is to **to explore methods for managing cloud resource scarcity** in a way that preserves both autonomy and timeliness. The thesis addresses this goal from different perspectives and consists of an introduction to the fields of cloud federations, scheduling, and quality-elasticity together with the following papers:

Paper I E. Elmroth and L. Larsson. Interfaces for Placement, Migration, and Monitoring of Virtual Machines in Federated Clouds. *Eighth International Conference on Grid and Cooperative Computing (GCC)*, IEEE, pp. 253–260, 2009.

Paper II L. Larsson, D. Henriksson, and E. Elmroth. Scheduling and Monitoring of Internally Structured Services in Cloud federations. *IEEE Symposium on Computers and Communications (ISCC)*, IEEE, pp. 173–178, 2011.

Paper III D. Espling, L. Larsson, W. Li, J. Tordsson, and E. Elmroth. Modeling and Placement of Cloud Services with Internal Structure. *IEEE Transactions on Cloud Computing*, IEEE, Vol 4, No 4, pp. 429–439, 2016.

Paper IV L. Larsson, W. Tärneberg, C. Klein, and E. Elmroth. Quality-Elasticity: Improved Resource Utilization, Throughput, and Response Times via Adjusting Output Quality to Current Operating Conditions. *IEEE International Conference on Autonomic Computing (ICAC)*, IEEE, pp. 52–62, 2019.

Paper V L. Larsson, W. Tärneberg, C. Klein, E. Elmroth, and M. Kihl. L. Larsson, W. Tärneberg, C. Klein, E. Elmroth, and

	M. Kihl. *Software: Practice and Experience*, John Wiley & Sons Ltd, 2020.
Paper VI	L. Larsson, H. Gustafsson, C. Klein, and E. Elmroth. Decentralized Kubernetes Federation Control Plane. *Submitted*, 2020.
Paper VII	L. Larsson, W. Tärneberg, C. Klein, M. Kihl, and E. Elmroth. Towards Soft Circuit Breaking in Service Meshes via Application-agnostic Caching. *Submitted*, 2020.

In addition to the papers included in this thesis listed above, the following publications and patents arose from work conducted by Lars during his doctoral studies:

Paper VIII	B. Rochwerger, C. Vázquez, D. Breitgand, D. Hadas, M. Villari, P. Massonet, E. Levy, A. Galis, I. M. Llorente, R. S. Montero, Y. Wolfsthal, K. Nagin, L. Larsson, and F. Galán. An Architecture for Federated Cloud Computing. *Cloud Computing: Principles and Paradigms*, John Wiley & Sons, Inc., pp. 393–411, 2011.
Paper IX	J. Ruuskanen, H. Peng, A. Åkesson, L. Larsson and M. Kihl. FedApp: Research Sandbox for Application Orchestration with Kubernetes on the Next-Generation Cloud and Edge Computing Infrastructures. *Submitted*, 2020.
Patent I	E. Elmroth, P. Gardfjäll, J. Tordsson, A. Aley El Din Hassan, and L. Larsson. Method, Node and Computer Program for Enabling Automatic Adaptation of Resource Units. *EP2904491B1*, European Patent Office, 2017.
Patent II	H. Gustafsson and L. Larsson. Kubernetes Decentralized Cluster Federation. *Filed*, 2020.

Financial support for this work was provided in part by the European Community's Seventh Framework Programme (FP7/2001–2013) under grant agreements no. 215605 (RESERVOIR) and no. 257115 (OPTIMIS); the Swedish Government's strategic research project eSSENCE; and the Swedish Research Council under contract no. C0590801 (Cloud Control).

Acknowledgements

I am incredibly fortunate to have been in the cloud field almost since its very beginning. Erik Elmroth, my main supervisor, has made that journey possible in many ways — not just by obviously taking under his wing a PhD student, but also letting me gain industry experience in the company he and Johan Tordsson founded, Elastisys. I am so thankful for the many opportunities you have given me, and for always being both willing and solutions-oriented enough so we found ways of working together, no matter what else we had going on in our lives.

As I write these words, my next career challenge is to act as branch manager for the Elastisys office in Lund together with my co-supervisor for this thesis, Cristian Klein. Thank you for your support, insights, and how you have challenged me during these years! I very much look forward to this next step. And with so many of our former colleagues from the research group being employed by Elastisys in Umeå, it feels like we get to bring the best things with us, even if we have moved over a thousand kilometers closer to a less harsh climate.

During my many years of being associated in one way or another to the Department of Computing Science, I have made many friends and have met some truly remarkable people along the way. Since Umeå is also where I studied for my undergraduate studies beginning in 2004, and I started working at the department in 2006 as a teacher's assistant (*amanuensis*), the list of everybody I would like to name here is virtually endless. Even "just" listing the members that the research group has had over the years would be around 20 names! So while I may not write out everybody's names, I hope you know in your heart as much as I do that you have all helped shape me, personally as well as professionally (and to some extent physically, via fun floorball matches at IKSU).

I would like to give a special thank you to my former office-mate Daniel Bergström (formerly Espling and before that Henriksson) for all the great times at home and abroad. And to my Elastisys office-mate, Peter Gardfjäll, who taught me not just a lot about life, but also about being professional. P-O Östberg is someone I have not yet shared an office with, but instead of that, we have shared countless of great times out on the town and talked about life, the universe, and everything — you are one of my inspirations for choosing the academic path in the first place. And my dear friend Ahmed Ali-Eldin, with whom I can (and have!) talk for hours on end with about topics ranging from the most spiritual to the down to earth ones, such as the many joys and challenges of parenting. Thank you all, you have greatly impacted and enriched my life, and will continue to do so for many years to come.

Even after life had taken me, my wife, and our three kids to southern Sweden, and even after many years of work in the industry, Erik Elmroth still made it possible for me to continue my PhD studies from afar. Maria Kihl graciously invited me to have an office at the department of Electrical and Information Technology as a visitor to her group. And down here in the south, in the "academic farmer's village" that Carl von Linné called Lund, I met William Tärneberg. We immediately found our extensive common ground and have collaborated on most of my recent papers. Thank you for our long conversations about both personal and professional topics, and for all the support we have given each other. You inspire me. Further inspiration came via an internship at Ericsson Research, where I was given the great opportunity to teach a PhD-level cloud course together with Johan Eker, and discuss where the industry is headed in edge computing with, among others, Harald Gustafsson. Thank you to you and Joakim Persson for giving me this great insight into the future as shaped by a company that is essentially the innovation engine that keeps Sweden running smoothly.

With a large amount of friends rooting for me and cheering me on to finish my PhD studies, my largest support nevertheless comes from home. From my wonderful wife Anna and our exceptional and incredible three children: Tina, David, and Samuel. You all give me love, purpose, and direction in life. I don't have words big and meaningful enough to adequately explain how much you mean to me. My mother Ursula and my sister Lisa, thank you for enabling, supporting, and challenging me as I became interested in learning about computers from an early age. None of this would have been possible if it were not for you encouraging me, and for that I am forever grateful. To you I definitely owe much of my work ethic and exploratory mindset, i.e. the key personality traits required to both start and finish an ambitious project such as getting a PhD. Finally, I wish my father Bo was able to celebrate this occasion together with us — may he rest in peace.

Thank you to everyone that believed in me and have helped me see this whole process through. It's been one heck of a ride, and from start to finish it sure took a while, but I am so very happy to have finished it all. In all the ways that truly matter, I feel like we did it together.

Lars Larsson
Lund, August 2020

Contents

1 Introduction **1**
1.1 Research Goal and Objectives 2
1.2 Historical Context 3
1.3 Thesis Outline 4

2 Cloud and Edge Computing **5**
2.1 Utility Computing: Vision and Implementation 7
2.2 Virtualization of Resources 8
2.3 Forklifting to Cloud-Native: Cloud Paradigm Shifts 10
2.4 Cloud Perspectives: Everything as a Service 17
2.5 Edge Computing 18
2.6 Research Methods in Cloud and Edge Computing . . . 21
2.7 Ethical Aspects of Cloud Computing 23

3 Federated Cloud Infrastructure **27**
3.1 Early Vision of Cloud Federations 29
3.2 Current Implementation of Cloud Federations 31
3.3 Future Vision of Cloud Federations 33

4 Scheduling **35**
4.1 Scheduling as an Optimization Problem 36
4.2 Co-development of Scheduling and Cloud Computing 38
4.3 Scheduling Across Cloud Federations 41
4.4 Scheduling in Future Cloud and Edge Computing . . . 42

5 Quality Elasticity **45**
5.1 Approximate Computing Primer 47
5.2 Eventual Consistency in Databases 47
5.3 Adjusting Application Output Quality to Resource Availability 52

6 Summary of Contributions **55**
6.1 Outline of Contributions 55
6.2 Paper I 59
6.3 Paper II 60
6.4 Paper III 62
6.5 Paper IV 64
6.6 Paper V 66
6.7 Paper VI 68
6.8 Paper VII 70

7 Conclusion **73**

Bibliography **75**

Chapter 1

Introduction

Cloud computing is a nebulous term that is currently used to describe a large array of computational services. Unless stated otherwise, we view cloud computing through the *Infrastructure-as-a-Service* perspective, in which an *Infrastructure Provider* owns one or more physical data centers and rents parts of this infrastructure out to *Service Providers* in such a way that each Service Provider has the illusion of accessing a private and dynamically resizeable data center of their own, without the operational costs and complexities associated with maintaining physical hardware.

Infrastructure Providers market their services as offering access to infinite, pay-as-you-go access to compute, storage, and network infrastructure capacity that is available to all and limited only by the imagination —or credit card limits— of the Service Providers. Because no resource, computational or otherwise, can really be infinite, such marketing statements cannot be true. This thesis focuses on the region where marketing claims collide with reality, and shows how this collision can be softened by using new methods for *managing cloud resource scarcity*.

This is a *collection of articles* PhD thesis. The introductory chapters, the *kappa*, present the context of the papers comprising the main content of the thesis. These chapters make the case that cloud computing, as a technology, has evolved together with the expectations of cloud users, and that their influence over each other has led to the emergence of software that can truly be considered *cloud-native*. Such software is adapted to thrive symbiotically in cloud environments, able to benefit from features of the cloud and work around its limitations while simultaneously offering leeway for cloud optimization (a more concise and complete definition is provided on page 14).

1.1 Research Goal and Objectives

This thesis has a single research goal: **to explore methods for managing cloud resource scarcity**. This is achieved by (a) proposing ideas and research directions in position papers, and (b) quantifying the efficiency and/or applicability of novel concepts experimentally via simulation or practical implementation. The problem of managing resource scarcity is viewed through the complementary perspectives of Service and Infrastructure Providers, depending on which party is responsible for implementing the solution under consideration. These perspectives are reflected in the following research objectives:

RO1 To explore ways in which Service Providers can adapt their services such that they react to resource scarcity by adjusting output quality and therefore their momentary resource needs.

RO2 To explore ways in which Infrastructure Providers can, through collaboration or intelligent resource allocation, proactively avoid subjecting Service Providers to resource scarcity.

Some of the work combines these two perspectives to offer solutions based on collaboration between the Infrastructure Provider and Service Provider, embodying the co-evolution that gave rise to cloud-native software.

By achieving both these research objectives, the work presented here meets the overall research goal. All of the solutions proposed herein adhere to the following two guiding principles:

GP1 Preservation of autonomy: individual cloud Infrastructure Providers should be able to autonomously optimize use of their own infrastructure based on objectives such as minimizing power consumption or other operational expenditures.

GP2 Mitigation timeliness: resource scarcity mitigation, regardless of which party initiates it, should either happen very quickly upon detection of scarcity or be initiated ahead of time by predicting potential problems before they materialize.

Building on the founding principles of cloud and edge computing[1], the three pillars upon which the work in this thesis rests are:

(a) **federations** of autonomous but collaborating cloud and edge computing sites;

(b) **scheduling** of service components onto resources within a cloud or edge site and across a federation thereof; and

(c) **quality-elasticity**, in which service response quality is adjusted based on the quantity of resources currently available.

1.2 Historical Context

This thesis has been many years in the making: I began work on it in 2009, practically at the dawn of cloud computing, and completed it in 2020. The research work presented herein was interspersed with work in the industry as a cloud architect at both the startup company and large enterprise level, and with an internship at Ericsson as a visiting researcher[2]. These different perspectives have taught many lessons and granted a rather unique perspective on the field's evolution. The 11 year duration is therefore an asset: the value of the earlier papers (published between 2009 and 2016) to the cloud research community has been proven by the test of time.

Readers of this kappa can therefore look forward to a personally experienced historical perspective on how technology has enabled change in both software and the minds of cloud users, resulting in today's

[1]For the purposes of this introduction, edge computing can be seen as a variant of cloud computing in which data centers are physically closer to end users. See Chapter 2 for details.

[2]It cannot be stated clearly enough that any views expressed in this thesis are those of the author alone, and are not in any way indicative of those held by any past or present employer or granter of internships.

cloud-native world where cloud applications are estimated to account for 90% of all mobile traffic [VT16].

1.3 Thesis Outline

The remainder of this thesis is structured as follows. Chapter 2 introduces the reader to cloud and edge computing, and explains how technological evolution helped shape the software design and deployment paradigms that we consider obvious today. It also introduces the methodologies used to perform research in this field (Section 2.6) and discusses the ethical ramifications of pervasive cloud computing (Section 2.7). Chapters 3, 4, and 5 describe the three pillars of this thesis – federated cloud infrastructures, scheduling, and quality elasticity – in sufficient detail to let the reader understand the context of the thesis. They also demonstrate the co-evolution between cloud and software that has happened within these fields from the dawn of cloud computing to the present day. A summary of the papers, their key scientific contributions, and notes about authorship contributions to each is given in Chapter 6. Chapter 7 presents some concluding thoughts; it is followed by the seven papers included in the thesis (see the Preface for additional papers and patents produced during the author's studies).

Chapter 2

Cloud and Edge Computing

This chapter presents a general introduction to cloud computing, briefly reviews its evolution from its inception in the early 2000s, and describes the branching off into edge computing that began a few years ago. The focus of the chapter is to show how software has evolved together with the cloud, with evolutionary steps in each enabling further evolution in the other, ultimately breeding software that can be deemed *cloud-native*. Following a general discussion of what cloud and edge computing are, we dive into selected topics related to cloud and edge computing, which are presented in an introductory manner.

First, we examine the vision and implementation of Utility Computing — the idea that computing power should be delivered in a fashion similar to, e.g., electrical power, and that it should be possible to let applications dynamically consume computing power according to need and subsequently pay per use. Second, we briefly cover the technological evolution in the area of virtualization that made cloud computing possible. Third, we take a historical perspective and discuss the paradigm shifts that have driven the co-evolution of the cloud itself and the software deployed onto it. Fourth, we describe the different perspectives through which cloud computing is commonly viewed, to help set the reader's expectations about the scope of this thesis. Fifth, we discuss different methods commonly used within cloud computing research. Sixth and finally, we briefly discuss some ethical ramifications of cloud computing because it has become a pervasive model for performing computation today.

However, before getting into those topics, we must first establish a common high-level understanding of what cloud computing is. The US National Institute of Standards and Technology (NIST) succinctly defines cloud computing as follows [MG11]:

Cloud computing

> Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or Service Provider interaction.

Stakeholders

In this thesis, we refer to the owners of the data centers that constitute the "shared pool of configurable computing resources" from the definition as *Infrastructure Providers* (IPs). Consumers of cloud resources are *Service Providers* (SPs). They use and pay for the infrastructure to provide some service (or application) to their *end users*. These terms denote the stakeholders and are used throughout the thesis.

Edge computing

Edge computing currently lacks a similarly broadly accepted definition (but see the work of [You+19] for a comprehensive survey of proposed definitions and a very good new one). For the purposes of this thesis, it is sufficient to note that edge computing is a form of cloud computing that uses similar technology and satisfies the above definition but with the following key differences: (a) edge data centers are deployed closer to the network edge, i.e. closer to end users, and are therefore more geographically dispersed, whereas cloud data centers serve Service Providers on a continental scale; (b) there are orders of magnitude more edge locations than centralized cloud locations; and (c) resource availability is orders of magnitude lower at individual edge locations than at individual cloud locations. Having established that edge computing is an evolution of cloud computing, we can turn to their shared history and the vision they both aim to implement: utility computing.

2.1 Utility Computing: Vision and Implementation

From a high-level perspective, cloud computing is the most recent implementation of the *utility computing* vision. John McCarthy is quoted as having stated the following during his MIT Centennial speech in 1961 [Gar99]:

Utility computing

> If computers of the kind I have advocated become the computers of the future, then computing may someday be organized as a public utility just as the telephone system is a public utility[…] The computer utility could become the basis of a new and important industry.

Two aspects of public utilities are important to highlight as they relate to cloud and edge computing: (a) utilities are typically provided and managed by a third party, and (b) the payment model for utilities is typically that you pay according to use. Economies of scale dictate that the larger a utility provider, the lower the amortized average per-unit cost. There is thus a clear incentive to offer the utility from the largest possible source, much like fresh water purification is most economically done at large plants serving entire cities rather than by using filters for each individual cup of water. The same holds true for computer-related utilities, which is why large cloud Infrastructure Providers operate large data centers around the globe with millions of servers [Yao+12].

Because some consumers require vastly more of a utility than others, a flat rate is inappropriate: single-person households in apartments and large families with swimming pools should pay fairly according to their use of water. Like with water, very large consumers of utilities often have special framework agreements in place with a utility provider, giving them lower rates as an incentive to stay loyal to the provider. Cloud computing has a pay-as-you-go model for computational resources, often now with a per-minute or per-second resolution, and like with water, large consumers such as the streaming video service Netflix will pay less per transferred byte than a more typical consumer.

At the core of cloud computing is a simple trade between a party that owns computers and another party willing to trade money for access to them. But that is not new — time-sharing systems have existed since the dawn of large-scale computing itself [CMD62]. What makes cloud computing different from earlier ways of trading money for compute power is in the underlying technology and the *expectations* that Service Providers can reasonably have because of it.

Grid computing

Before cloud computing, Grid computing was the predominant implementation of the utility computing vision. Although early papers [Fos02; Fos03; Fos+08] were optimistic and likened Grid computing to the power grid with its "plug in and consume as needed" ease of use, the reality was a cumbersome mishmash of technologies with an overall user experience tolerated only by academics needing access to high-performance computing (HPC) resources. Grid computing implementations, true to their HPC heritage, primarily ran programs in non-interactive batch mode with poor inter-process isolation. Programs had to be tailored specifically to the operating system and set of libraries offered by the specific machines in a specific Grid. Attempting to use non-standard libraries was cumbersome, to describe the user experience mildly. Thus, in the vernacular of 2020, utility computing as a field was ripe for disruption.

2.2 Virtualization of Resources

Rather than making applications conform to limitations imposed by the underlying (Grid computing) platform, application developers wanted to specify their own custom software installations, including the operating system, supporting libraries, and specific applications. This is precisely what (hardware) virtualization offers: it lets a computer process simulate multiple "virtual machines" (VMs), allowing "guest" operating systems to be installed and share access to the underlying hardware via the host operating system. Despite the possible utility of virtualization and the fact that it is an idea with a long history (dating back to about the 1970s [MS70]), its adoption was held back by poor performance: it was simply too slow to simulate an entire machine in software alone.

Virtual machine

However, around 2005 and 2006, the major CPU vendors Intel and AMD started offering CPUs with hardware-assisted virtualization support. This enabled full-system virtualization with acceptable performance degradation (in particular, I/O was slow at the time [Dre08]; later advances in hard- and software further reduced the degree of degradation [Gor+12]). Instead of running processes on a shared operating system and suffering from poor isolation, virtualization allowed developers to provision virtual machines with full operating systems and free choice of supporting libraries to suit their desires and needs.

Businesses benefited greatly from virtualization because it reduced their operational expenditures, even before cloud computing became commonplace. It is well-known that average server resource utilization is typically in the 15–20% range [Vog08; BH07]. Using virtualization to consolidate several server functionalities onto a smaller quantity of physical resources reduced the need to host, manage, and maintain needlessly large private data centers. With the added ability to export and import virtual machines from one physical computer to another, resource use could be dynamically optimized. Because they are merely abstract constructs in software, virtual machines could eventually be migrated between physical machines even during active use [Nel09; Jin+09; Svä+11] (so-called live migration), making it possible to seamlessly migrate away virtual machines if the underlying hardware became too overloaded or needed replacement.

Migration of VMs

These abilities were immense boons to productivity. Eventually, businesses were able to offload even the reduced management of physical machines to large data centers that promised to do the menial tasks more cheaply than in-house system administrators and IT personnel. Thus, the idea of the cloud was born. Virtualization and the on-demand provisioning of virtual infrastructure that it enabled made cloud computing "convenient" (see definition on Page 6), which was a key point of differentiation from competing alternatives such as Grid computing.

Despite their convenience, virtual machines alone were not enough to enable the cloud revolution to take off; efficient networking and storage were also required. Virtualization played a key role here too. Software-defined networking (SDN) is a form of virtualization for networking equipment [McK+08], in which the control plane (the making of logical decisions about how to route traffic) and data plane (the actual physical traffic routing) are separated. Before SDN, networking equipment typically embodied both functionalities and was not designed for

Software-defined networking

global optimization. In contrast, SDN enabled highly efficient virtual networks that could adapt as virtual machines came into and out of existence or migrated across physical machines [JP13].

Storage virtualization Storage virtualization, whereby physical storage devices are pooled into logical ones and made available across the network, enabled the logical decoupling of data from the physical media on which it is stored. Storage services thus made it possible to attach what appears to be a physical block device to a virtual machine, when in reality the data is served over the network from a fully redundant storage solution with several backing devices. Such virtual block devices can be instantly transferred from one virtual machine to another.

2.3 Forklifting to Cloud-Native: Cloud Paradigm Shifts

With the basic technological building blocks in place thanks to virtualization of computation, networks, and storage, cloud computing was poised to revolutionize the way we perform computing at scale. However, early adopters were still bound by old paradigms and thus limited in their approach to the technology. Similarly, the technology was limited by the prevailing expectations. Several paradigm shifts were therefore needed to reach our current position.

Before the cloud, the most common solution to resource scarcity problems was to get a faster machine with more memory and hope the problem subsided. This was simultaneously expensive, complicated, and time consuming because it involved physical machines that had to be managed on-site. Because software at the time often mixed data processing with data storage, this solution also seemed natural — high resource requirements due to high levels of data processing or storage were met by getting a newer, faster, and "bigger" computer.

Forklifting The first major use case for cloud computing was what became known as "forklifting" [Var10] , in which a server that used to be deployed and managed on-site by local system administrators was replaced with a virtual machine provisioned at a cloud Infrastructure Provider. Companies that did this saved money by not having to house or replace physical hardware, and by making system administration easier to centralize and outsource because the virtual machines were accessed over the Internet.

Like their old physical counterparts, forklifted virtual machines were initially still regarded as special machines that were cared for, managed, and maintained individually. But virtualization and the pay-as-you-go pricing model offered other tantalizing new opportunities: additional or larger virtual machines could be obtained with just an API (Application Programming Interface) call, without having to go through the week-long procurement procedures required for physical hardware. Thus even the old approach to managing resource scarcity — getting a larger virtual machine — was greatly accelerated. Doing this with minimal service disruption became known as *vertical elasticity*, and was initially achieved by shutting down and replacing the old virtual machine while keeping its storage intact. Later, seamless vertical elasticity became possible, using techniques such as memory ballooning [CLC13]. This solution to the resource scarcity problem did not make the best use of the cloud's capabilities, but it was at least faster than its physical counterpart. Also, benefits such as snapshots of data stored in virtualized storage services helped system administrators do their job.

Vertical elasticity

However, a virtual machine cannot exceed in memory the size of the physical machine upon which it is deployed without very obvious performance penalties due to factors such as memory swapping. Consequently, there was and always will be a limit on the amount of additional resources that can be granted to applications through vertical elasticity. But the cloud offered an alternative way of scaling: obtaining *more* virtual machines. To exploit this capability, software had to be designed without strong coupling between data and its processing. With applications designed in this way, resource scarcity could be solved by obtaining more virtual machines for processing instead of growing a select handful in size. Scaling out rather than up in this way became known as *horizontal elasticity*. Figure 2.1 shows the two ways of scaling and the limitations of vertical elasticity.

Horizontal elasticity

By separating data from its processing and relying on the cloud to supply seemingly infinite numbers of virtual machines, system administrators gained access to infrastructure that could grow and shrink according to need. Rather than the replicated database systems of

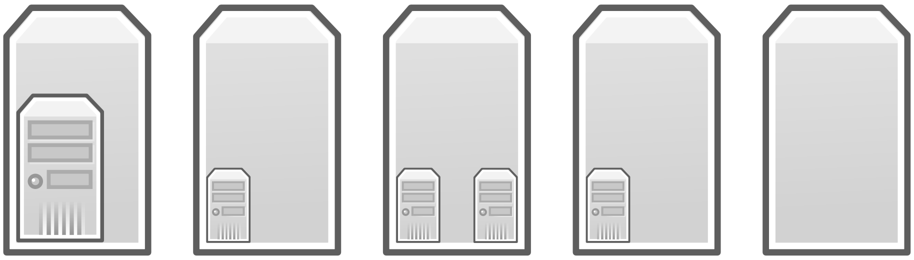

Figure 2.1: Two types of scaling offered in cloud computing by using virtualization, exemplified using a virtual machine that needs 4 times its initial capacity. In vertical elasticity (leftmost server), a virtual machine with 4 times the capacity of the original is requested. Scaling vertically is therefore limited by the capacity of the physical server that hosts the virtual machine. Horizontal elasticity (the four rightmost servers), on the other hand, requests 4 virtual machines (possibly from different servers), and scaling is thus limited only by the number of servers, of which there could be millions [Yao+12].

yesteryear that relied on active replication between a single primary server and a number of secondary ones, peer-to-peer database systems such as Cassandra and MongoDB could offer nearly linear scalability (see Section 5.2). The move to horizontal scaling with almost linear performance improvement was the first step toward cloud-native software.

However, managing an orders of magnitude larger fleet of virtual machines soon became cumbersome to system administrators. System administrators also started questioning the value of managing individual virtual machines and upgrading them in concert when virtual machines could simply be terminated and replaced for the same financial cost (paid to the cloud Infrastructure Provider). This led to the *pets vs. cattle* paradigm shift [Bia16] whereby servers stopped being treated as individuals that needed to be cared for (pets) and were instead seen as mere numbered units worth only the function they presently provided (cattle). *Configuration management* software, such as Puppet, Chef, and Ansible, was developed to bring a newly started server to a designated target state. This process could be initiated using a stock base image with just a standard operating system install. However, this could be lengthy and was only as bug-free as other software made by developers.

Pets vs. cattle

Configuration management

The process was significantly accelerated and made less error-prone by using custom virtual machine images containing the initial hard drive contents upon startup. In this way, servers were brought up with known-good software versions in (mostly) ready-made virtual machine images, requiring only runtime configuration to make the virtual machines operational. Target state descriptions were expressed in domain-specific and text-based languages, and could be stored in version controlled repositories.

The simplification of (virtual) infrastructure management achieved by decommissioning unsuitable VMs and replacing them with known-good new instances together with the introduction of software tools enabling infrastructure to be managed in a way that came naturally to developers led to the commingling of development and operations, which is now called *DevOps*. Tools for managing infrastructure in this code-like fashion popularized the *Infrastructure as Code* movement, which held that any manual system administration was a bad practice (an anti-pattern) because it relied on individual expertise, and that recovering an entire system in case of disaster presented a high risk of error due to human fallibility. Tools such as Terraform and Packer provided much-needed functionality enabling the implementation of the Infrastructure as Code approach, especially as Service Providers started using larger parts of the ever-growing infrastructure services offered by Infrastructure Providers. These enabling functionalities included domain name management, load balancing tools, and virtual private networks.

DevOps

Infrastructure as Code

The movement to configuration management of fleets of easily replaceable virtual machines was the second large step toward making software cloud-native. Because it made (subsets of) virtual machines less unique, and hence much more replaceable, it gave Infrastructure Providers an opportunity to use available resources more efficiently and to optimize scheduling by terminating machines at will so as to optimize the use of the infrastructure. This led to the emergence of *spot instances*, ephemeral virtual machines that can be terminated with little warning but are offered at a large discount. Extensive economic research has been conducted to determine how best to use spot instances [MD11; YKA10; YAK12; TYL12; KS15], but without the supporting technology that made it feasible in the first place, the concept of ephemeral virtual

Spot instances

machines would never have seen the light of day. The intended use cases and user expectations of spot instances differ markedly from those of pet servers forklifted into the cloud. For Infrastructure Providers, spot instances offered a way to recoup some of the losses incurred by implementing the illusion of "infinite resource availability", which requires large amounts of spare capacity. Spot instances make profitable use of this spare capacity when it would otherwise be idle, at essentially no additional cost to the Infrastructure Provider.

Stateful vs. stateless

A key driver of the paradigm shift in favor of replaceable virtual machines was the realization that the *stateful* components of an application (e.g., databases) should be separate from its *stateless* (data processing) components. Stateless components can then be independently and automatically scaled up or down based on need and resource availability.

Stateful components can be further subdivided into those requiring strict consistency and those that merely require a state that is eventually consistent in a quality-elastic way (Chapter 5). Lowering expectations of state consistency when designing applications enables both the applications and the cloud to make better use of currently available resources without causing errors or unacceptably poor performance.

Although the concept of cloud-native software has been discussed extensively, it lacks a universally accepted definition. Definitions have been proposed based on both experience [Tof+17] and literature surveys [KQ17]. Building on the brief discussion of the co-evolution of clouds and software in this section, we propose that cloud-native software be defined as follows:

Definition 1 (Cloud-native software) *Software that has adapted to the cloud, and to which the cloud itself has adapted such that it displays (a)* resilience *to failures in hard- and software, and (b)* elasticity *in its ability to scale according to resource demand by drawing on features of the underlying virtualized cloud infrastructure; and (c)* compartmentalization *into separate task-specific deployment units, enabling task-appropriate resilience and elasticity operations to occur.*

Definition 1 is consistent with previously suggested definitions [Tof+17; KQ17] and adds the explicit requirement that software should be compartmentalized into task-specific units.

Industrial groups such as the Cloud Native Computing Foundation and Red Hat have recently argued that compartmentalization *technology* is a crucial aspect of cloud-native software [Red18; Com18], specifically dictating the use of (software) containers over virtual machines. Containers (specifically, Linux containers) leverage Linux cgroups to provide a greater degree of inter-process isolation than was possible at the operating system level in Linux [Mer14]. Containerized processes are executed by the host machine's operating system kernel and enjoy improved resource limiting, prioritization, and control [Ros14]. This provides direct access to hardware without virtualization, eliminating the overhead of virtualization [Joy15] (although it should be noted that the magnitude of this overhead has been greatly reduced by improved virtualization support in both hard- and software). According to our definition, the specific compartmentalization technology used is not really a key determinant of whether a given piece of software can be considered cloud-native; it should not matter whether compartmentalization is achieved using a particular type of Linux container or a VM.

Containers

Maturing cloud infrastructure and software let Infrastructure Providers invent new service delivery models, as exemplified by the introduction of AWS Lambda in 2014. This model embodies the notion of *serverless computing* , in which applications serving requests are logically split into two parts: (a) an ever-present API endpoint that listens for incoming requests, and (b) any number of request processors that are started on-demand to handle the requests. Requests that reach the API endpoint are queued (if needed) before being routed to a request processor. This model is described as serverless because Service Providers need not manage any of the underlying infrastructure required to implement this functionality. Instead, it is the responsibility of the cloud Infrastructure Provider to ensure rapid invocation of the request processor software and to automatically (horizontally) scale the number of such processors to meet current demand. Crucially, if there is no demand, the number of request processors can be scaled down to zero.

Serverless

Request processors are typically supplied as stateless functions, written in whichever programming languages are supported by the underlying cloud runtime environment used by the cloud Infrastructure Provider. Alternative open source implementations of similar technologies, e.g., OpenFaaS, allow for any containerized software to be supplied. Regardless of the underlying technology, successful implementation of the serverless or Function-as-a-Service (FaaS) paradigm requires addressing the problem of cold starts: starting a new request processor necessarily involves scheduling and instantiating the processor itself, both of which take time. A common technique for addressing the cold start problem is to keep each request processor instantiated for a while after it has served a request, based on the assumption that once a function has been invoked, it may soon be invoked again [Man+18; Wan+18; Moh+19].

As with the other enabling technologies discussed above, the concept of starting a process upon receiving an incoming request is not new. The UNIX Internet daemon (inetd) from 1986 and the Common Gateway Interface (CGI) from 1993 were both popular and widespread implementations of the same idea, albeit on a much smaller scale. What makes the serverless model unique in cloud computing is that it lets the cloud Infrastructure Provider both (a) accept responsibility for running what are essentially processes, turning the cloud infrastructure into a large operating system; and (b) support a large (or even larger) number of customers concurrently on the same hardware, because typical servers and the services they host are only infrequently used [Vog08]. Infrastructure Providers thus gain another way to add value to "raw" infrastructure, and to dynamically meet the resource needs of larger numbers of Service Providers using existing infrastructure.

As shown by the preceding discussion, over the decade and a half of its existence, we have seen the cloud evolve from a way to outsource maintenance of a physical private data center to a major utility provider into a foundation for performing computing at scale. The cloud has enabled software to evolve in new ways to best make use of the available infrastructure, and developments in software have in turn allowed cloud infrastructure to evolve in new ways, for example by offering spot instances. Many of the concepts involved — virtualization, eventual consistency, and configuration management — were not new. However, their confluence, the ways in which improvements in one were leveraged by others, and the timing of their advances and improvements allowed cloud-native co-evolution to occur.

2.4 Cloud Perspectives: Everything as a Service

Cloud computing offers on-demand access to resources. But what is a resource: virtual infrastructure, platforms upon which applications can be built, or entire applications? Thanks to a combination of strong marketing and the success of cloud computing in revolutionizing the ways in which computational services are offered and consumed, all three responses are (confusingly) valid.

Several perspectives have been used to clarify the overall concept of cloud computing. The NIST cloud definition report highlighted the following three [MG11]:

- **Infrastructure-as-a-Service** (IaaS), in which Infrastructure Providers offer metered access to virtual infrastructure (computing, networking, storage) to be consumed by Service Providers.

IaaS
PaaS
SaaS

- **Platform-as-a-Service** (PaaS), in which metered access to software platforms consisting of generic components such as databases, message queues, or email sending services is offered. Service Providers leverage these to focus on developing their applications and to reduce operational overhead otherwise required to maintain these platform components.

- **Software-as-a-Service** (SaaS), in which metered access to entire applications is offered to end users as in the case of Office365 and Google's G Suite.

In this thesis, we view cloud computing from the Infrastructure-as-a-Service perspective. Research adopting this perspective typically focuses on how to best offer infrastructure to Service Providers in order to meet their expectations and enable new paradigms, and on how to optimize the use of physical infrastructure [Fra12; Fra+12; SH11; Roy+15; Krz+18].

Developments in cloud and edge computing have led to the emergence of new "as-a-Service" offerings, many of which can be considered to represent niches of the original three perspectives. The widespread use of containers rather than virtual machines has prompted some to question whether Service Providers should have to manage all the virtual infrastructure required to host the clusters of computers upon which containers are deployed. **Containers-as-a-Service** offerings such as AWS Fargate[1] remove this burden, and thus occupy a niche somewhere between Infrastructure- and Platform-as-a-Service. **Function-as-a-Service** offerings instead give Service Providers automatically scalable and purpose-built Platform-as-a-Service environments tailored to the needs and particularities of serverless computing. Service Providers then only need to upload functions, mere snippets of code, and all infrastructure and connections to platform services are handled for them by the Infrastructure Provider.

As shown in the previous section and discussed at greater length in Chapter 4, the ability of Infrastructure Providers to optimize the use of their infrastructure is increased by taking on a greater part of the responsibility for deciding how applications are executed. It also means that the value, and hence the price to be paid by Service Providers, increases. We can therefore assume that more cloud-native service offerings will appear as the profitability of offering cloud infrastructure increases. Perhaps this is most clearly demonstrated by how telecommunication companies are now getting into this area of business by offering edge computing capabilities.

2.5 Edge Computing

Edge computing can be regarded as a "new generation" form of cloud computing that uses similar technologies and business models. As mentioned in the beginning of this chapter, there is no commonly accepted definition of edge computing. In their comprehensive survey, Yousefpour et al. regard edge computing as a proper subset of cloud computing [You+19, Figure 4].

What differentiates edge computing from cloud computing is that computational resources are allocated closer to end user equipment (mobile user equipment, computers, sensors, etc.) in networking terms: they may be separated by only a single network *hop* to a base station

[1] `https://docs.aws.amazon.com/AmazonECS/latest/userguide/what-is-fargate.html`

or an edge node deployed to service a group of households or a single Industry 4.0 factory plant [You+19].

Edge computing properties

This proximity has several effects when compared to general cloud computing, where the data center may be half a continent away from the end user: (a) last-mile latency to end users is reduced; (b) bandwidth pressure on backbone networks is decreased by processing data at the edge, which reduces the need to transfer raw data to centralized clouds; (c) computational and storage resources are more scarce because edge locations by definition are smaller than cloud data centers, which increases per-resource costs to Service Providers; and (d) operational costs for Infrastructure Providers *may* be higher because they cannot exploit economies of scale when managing the underlying hardware as efficiently as with large data centers.

Edge computing use cases

By virtue of its proximity to end users, edge computing can offer both lower latencies to end users and higher bandwidth availability than centralized cloud infrastructure. This benefits use cases that cloud computing data centers struggle to properly support [MB17; Shi+16], such as autonomous vehicles [Bye17; CJC19], augmented reality [SRS17; ES18], collecting and processing massive amounts of data from Internet of Things (IoT) applications [PDT18] (Cisco estimates that IoT sensors will number 23 billion by 2023 [Cis18]), and computer-assisted healthcare [Bye17].

Edge computing encompasses infrastructure belonging to both cloud and telecommunication providers. To get a sense of the numbers involved from the point of view of a cloud provider, Google states that as of June 2020, its infrastructure consists of 23 cloud regions (with a total of 73 availability zones) and 144 edge cloud sites[2]. Amazon Web Services has a similar size. On the other hand, in Germany alone, Deutsche Telekom will reportedly have some 36,000 base stations nationwide by 2021[3]. Whether all or just a fraction of these will offer computational infrastructure as edge sites remains to be seen, but the potential and sheer scale of these infrastructures makes it clear that edge computing presents a new set of challenges and opportunities to researchers and engineers.

[2]https://cloud.google.com/about/locations

[3]https://www.telekom.com/en/media/media-information/archive/300-new-lte-mobile-base-stations-574106

The cost per resource unit is inherently higher in edge computing than in centralized clouds, because edge computing is less able to make use of large economies of scale at individual operating locations. However, these increased operational costs may be offset by the fact that telecommunication networks, *regardless* of edge computing, require base stations with a certain amount of hardware that must be maintained. Increasing the amount of physical hardware in edge locations does not linearly increase the cost of operating it. Moreover, lower latency and reduced backbone network traffic are so hugely desirable for the aforementioned use cases and applications that higher per-resource costs and resource scarcity are acceptable trade-offs.

Edge computing derives from cloud computing and thus shares its historical roots. However, the concept of increased proximity to end users has a unique history of its own. The oldest form of edge computing is the content delivery network (CDN), which offered geographically dispersed static file hosting in proximity to end users. CDNs thus combined storage and network resources. When dynamic content became the norm for web pages, CDN providers such as Akamai developed the Edge Side Includes standard in 2001 [Tsi+01], which allowed for a limited amount of conditional logic to be performed on the edge to reduce the load on upstream (origin) web servers and avoid needless backbone network traffic (e.g., to only show a certain part of a web page to authenticated users). In more recent years, some CDN providers including the current market leader (Cloudflare) have started also offering general purpose computational resources at their edge locations (e.g., Cloudflare Workers).

Content delivery network

Edge and cloud computing are hence complementary rather than competing technologies; high bandwidth and low latency data processing at the edge will likely generate some results that must be stored and processed centrally in the cloud. Therefore, effective combined use of the edge and the cloud requires a software deployment that is mindful of the interconnecting network and its impact on overall service performance. Research has shown that choosing the right deployment split between edge and cloud deployments is vital [Ngu+19; McC+19].

Split deployment

The evolution of cloud and edge computing is important here because it affects all three pillars of this thesis: edge federations are several orders of magnitude more numerous those of cloud Infrastructure Providers, edge computing enables schedulers to scale down to zero and make rapid re-scheduling decisions, and quality elastic adaptations can be used to overcome the inherent resource scarcity of edge locations. These issues are all discussed at greater length in the upcoming chapters.

2.6 Research Methods in Cloud and Edge Computing

As the previous sections show, cloud computing facilities come in many different shapes and sizes, with different levels of resource availability. Consequently, the cloud research community relies on a range of common research methodologies to examine different research topics:

- **Simulations**, in which relevant parts of the cloud or edge infrastructure are simulated using some model. Experience shows that such simulations are particularly common for exploratory work in new fields because simulation makes it easier to focus on the novel aspects of the scientific contribution without having to dive deeply into implementation details. Alternatively, the systems under study may simply not exist yet (for instance when examining future generation infrastructure or computing systems) even though their properties are known or can be reasoned about in advance. It may also be the case that large-scale infrastructure is unavailable to researchers. Simulations are used in two of the papers included in this thesis, **Paper III** and **Paper IV**.

- **Data set or system trace analysis**, in which a system of interest has been deployed for a set of experiments or a period of time and its behaviors are analyzed offline or used to gain insight into the behavior of real-world applications. Google research typically disseminates results based on systems that have been run in production use for months or years, such as Map-Reduce [DG08], Chubby [Bur06], and Bigtable [Cha+06]. Additionally, some public data sets are available to support reproducible research. Two deeply influential data sets of this type originate from the FIFA 1998 World Cup web site [AJ99] and Google's cluster usage traces [RWH11; Wil11] from 2011. Both have been used extensively in cloud research, and the Google cluster-usage traces from 2019 [Wil20a; Wil20b] are sure to be similarly popular in the coming years.

- **Private cloud implementations**, in which the researcher has access to a privately owned set of physical hardware and uses it to conduct experiments. This has good and bad aspects: the generality of the results obtained may be questionable because every physical hardware configuration is unique and they are rarely well-documented, and differences in software configuration may also profoundly affect results. These issues are discussed in **Paper V**, which presents work done on a privately owned cloud that was not operated by the authors, necessitating creativity in working around issues arising from particularities of the cloud infrastructure. In more typical cases, where the entire physical hardware is under the researchers' control, the advantage of a private cloud is that full-stack observability is possible. However, this comes with a large additional system administration burden, and thus carries an increased risk of mis-configured systems impacting research results.
- **Public cloud implementations**, in which experiments are deployed on cloud or edge infrastructure belonging to public cloud vendors, such as Amazon Web Services, Google Cloud Platform, or Microsoft Azure. Public clouds present the opposite trade-off to private ones: full-stack observability is not possible, but the results are more reproducible because other researchers have access to the same experimental environment.

Many researchers have profiled the performance of public clouds [Ost+10; IYE11; Ios+11; LC16]. These studies treat the public cloud essentially like a black box, and observe it carefully in the same way that a natural scientist would study a natural phenomenon. Work of this sort is important because it informs business decisions about which cloud vendor to use, and also helps researchers understand the systems upon which their experiments are deployed. However, underlying changes to hard- or software can hugely affect the contemporary validity of such results [Pro+18]. To at least partially overcome this problem, the Standard Performance Evaluation Corporation (SPEC) releases a standardized cloud IaaS benchmark and publishes guidance on using it to perform repeatable scientific measurements [Pap+19].

To circumvent the problem of performance variability entirely (drawing inspiration from the difficulties described in **Paper V**), **Paper VII** presents a different approach. Rather than focusing on performance measurements, we instead count the number of messages transmitted over the network. The latency per request and the speed of processing requests are thus regarded merely as implementation-specific details. This is a common approach in distributed systems research, where algorithmic efficiency is evaluated based on how many messages need to be transmitted.

2.7 Ethical Aspects of Cloud Computing

Given the massive success of cloud computing and its key role in much of society's digital evolution, it behooves everybody in the field to consider the ethical ramifications of our technological advances. Although cloud computing is a major technological enabler of our large and increasing societal reliance upon pervasive computing, highly cited works outlining cloud research agendas such as [BCR09; Vou09; VB18] do not discuss any ethical considerations, preferring instead to focus on outstanding technological challenges. Some technological *naïveté* can be excused in older works of this type: cloud computing has put a previously unimaginable amount of data storage and processing power into the hands of anyone with a sufficiently large budget. Before cloud computing, nobody could reasonably believe that it could be economically viable to collect and process data in the way we do routinely today. However, while we clearly *can* (and did) build globally accessible software services and use cloud computing to process data from mobile smartphones and myriad Internet of Things devices, that does not mean that we unquestionably *should*.

Societal reliance on digital services has increased continuously in parallel with their explosive growth, which began with digitalization in the 1990s and was further fueled by the advent of smartphones in the late 2000s. Additionally, while smartphone vendors make great efforts to lock end users into a particular device family or line of operating systems, those users display stronger loyalty to the cloud-backed services that they trust with their data [PSK15]. This is because while a physical device may be replaced every few years, an email address or social media account will often survive several generations of devices used to access them. Thus, as important as smartphones are in today's society, the cloud services that power them are arguably more important.

As end users actively entrust more of their personal information to cloud Service Providers, questions regarding the legal jurisdictions governing the storage, processing, and usage of said data remain poorly explored and also poorly understood by many practitioners [KMZ15]. Legislation may require that data storage and processing be done in a certain country. But if either Infrastructure and Service Providers are subject to, e.g., US law, can such a guarantee be convincingly made for, e.g., German data?

Machine learning (ML) and artificial intelligence (AI) currently consume large amounts of computational resources, and cloud computing is often used to provide the infrastructure required to handle such workloads. Access to cheap computational resources has granted ML and AI a new wave of popularity by breaking the resource scarcity barriers that previously held them back [HK19]. This access to vast amounts of data processing power at relatively low cost, combined with ML and AI tools for processing Big Data datasets, has significant ethical ramifications and a profound societal impact for which we currently lack suitable regulation [Pag+19; Bar+20; Flo+18]. Companies have seized on this state of affairs as a business opportunity, giving rise to the *surveillance economy*, in which data about people is pervasively collected, processed, and monetized for financial gain, e.g., through advertising. Shoshana Zuboff states the following in her 2019 book "The Age of Surveillance Capitalism" [Zub19]:

> [Surveillance capitalism] unilaterally claims human experience as free raw material for translation into behavioral data [which] are declared as a proprietary behavioral surplus, fed into advanced manufacturing processes known as "machine intelligence", and fabricated into prediction

products that anticipate what you will do now, soon, and later.

The drive for data is so strong that many startup businesses spend investor money on implementing a loss-leader marketing strategy of undercutting the competition, actively losing money per customer in their initial phases to secure users' loyalty and thus obtain their data for future use [Ste20].

Once data is given to a company, users typically cannot influence what is done with it or with whom it is shared. Data provenance, i.e. tracking the points at which data may have been accessed and altered, would provide this information, but remains poorly understood. However, perhaps showcasing the engineer's inherent desire to address people problems with technological solutions, research in this field typically proposes various supposedly tamper-proof ways of collecting data and ensuring that modifications are attributed to the altering party [Lia+17; KW14; Sue+13; Asg+12], using encryption or other techniques. Approaching the issue from another angle, laws such as the EU General Data Protection Regulation (GDPR) force companies to explicitly list all partners with whom data may be shared. However, these lists are often so long and change so frequently in practice that a user cannot realistically comprehend them or be expected to contact all those companies to demand the deletion of all data they may have collected about the user.

Cloud computing is projected to account for 95% of all data center network traffic by 2021[4], and has become the technological foundation upon which all kinds of services, both agreeable and disagreeable, are built. On the more agreeable side of the spectrum, cloud computing has been a boon to democratizing software delivery. Many public cloud Infrastructure Providers offer free trials, and all that is required to make use of them and build a scalable business is the ability to start small and build from there with profitability in mind. As long as an entrepreneurial Service Provider has a computer with Internet access and a credit card, their physical location in the world and upfront economic resources are immaterial: they could build a globally

[4]According to the no longer available Cisco® Global Cloud Index (2016-2021) White Paper.

successful service on a shoestring budget with no initial investment in physical hardware and without moving to a major tech hub. The cloud is thus a means by which we can achieve our ends — whether those are good or bad is up to us as Service Providers.

Chapter 3

Federated Cloud Infrastructure

For certain use-cases such as services with geographically dispersed (global) user bases, or simply for redundancy, a single cloud site[1] is not enough. A single cloud site may be insufficient for service deployment because of the high latency-sensitivity of certain applications, legislative concerns about data storage or processing, redundancy requirements for fault-tolerance, or a need to achieve high availability. To support use-cases requiring multiple cloud sites, collaborations can be established between cloud sites belonging to a single cloud Infrastructure Provider or even between different cloud Infrastructure Providers.

This chapter begins by defining key properties of cloud federations. In accordance with the theme of this thesis, we show how federations have evolved over time in parallel with Service Provider expectations and cloud maturity, from the early days of the cloud (Section 3.1) to current implementations in cloud and edge computing (Section 3.2). Finally, we discuss the driving forces that have shaped cloud federations and show how they may affect the future of the field (Section 3.3).

[1]We use generic terms such as cloud *sites* here to refer to either regions or availability zones, which will be described more explicitly in Section 3.2.

In this thesis, we define cloud federations as follows:

Definition 2 (Cloud federation) *A collaboration, whether explicitly stipulated in predetermined agreements or implicitly enabled via compatible technology, between multiple (a)* independent *and (b)* autonomous *cloud sites.*

Independence Independence (non-reliance upon others) is the first key property according to our definition. Sites that are not independent of one-another can be regarded as a single site from the standpoint of the cloud even if they are separated geographically. Non-independence implies that a single cloud management software system manages all the physical hardware at the constituent sites.

Autonomy Autonomy (freedom to make own decisions) is the second the key property of our definition, and is so important to our work that preserving it is one of the guiding principles of this thesis (**GP1**). For reasons including cost-efficiency, failure isolation, and scalability it is imperative that each individual cloud site can self-optimize independently in terms of how applications are deployed on its infrastructure (for more details, see Chapter 4). This is particularly important if the cloud sites belong to different (and possibly competing) business entities.

Note that we use the term federation in reference to a collaboration between cloud sites that are maintained by Infrastructure Providers. Service Providers need not be aware that such federations exist, or may be only minimally aware. This increases the scope for Infrastructure Providers to optimize the use of their own infrastructure and to use that of other providers in the federation. In contrast, many Service Providers have, aided by technology that makes doing so easier, adopted a *multi-cloud strategy* that makes use of disparate cloud sites. While such a strategy can help achieve goals such as deploying applications closer to end users, failure isolation, and disaster avoidance, it falls outside of our definition of cloud federations. The main determinant of whether a collaboration can be considered a federation is thus whose responsibility it is to spread an application deployment across multiple cloud sites, and what it helps that party to achieve.

Multi-cloud strategy

3.1 Early Vision of Cloud Federations

Since the dawn of cloud computing, it has been clear that a single cloud site or provider cannot support all advanced use cases: the appearance of infinite resource availability cannot realistically be offered to Service Providers using the physical hardware of any single Infrastructure Provider. In addition, cloud federations offered interesting challenges and avenues for both research and product and service development. Some of these included the tantalizing possibility of making greater profits by accepting more Service Providers than local resource availability could support at peak usage, and offloading excess to other members of the federation [GGT10].

The EU-funded RESERVOIR FP7 project was an early pioneer in this field [Roc+09a; Roc+09b; Roc+11], and it was within the context of this project that the ideas motivating **Paper I**, **Paper II**, and **Paper III** were conceived. In the early days of cloud federations, Infrastructure Providers were focused on making federations possible and efficient on a technical level (**Paper I**), and on exploring what federation would mean for processes such as optimizing the scheduling of virtual machines across multiple cloud sites (**Paper II** and **Paper III**). Meanwhile, Service Providers were interested in using multiple cloud sites or providers because of the potential for cost savings [Tor+12] and to keep applications highly available [Vil+12].

Cloud management software

Cloud management software, i.e. software to turn physical hardware into "a cloud" for private use within a company or for research at a university, was an emerging field in the early days of cloud computing. Notable examples included Eucalyptus [Nur+09], CloudBus [BPV09], OpenNebula [MLM11], OpenStack [SAE12], and CloudStack [SS13]. These differed extensively under the proverbial hood and accordingly also presented incompatible management APIs to Service Providers.

It must be noted that the cloud landscape at the time was utterly dominated by Amazon Web Services. Private or research clouds using one of the cloud management software suites listed above existed both at companies and universities, but "the cloud" was for most intents and purposes entirely synonymous with Amazon Web Services. Therefore, all cloud management software suites also offered some additional APIs for compatibility, most notably APIs that were more-or-less compatible with Amazon Web Services Elastic Compute Cloud (AWS EC2). This supposed compatibility was merely superficial, and the Service Provider often had to use compatibility tools such as Apache JClouds [Ism+15] to use the native management APIs of each cloud Infrastructure Provider.

Superficial compatibility APIs aside, Service Providers at the time understood that a VM that had been deployed to one Infrastructure Provider would not easily be transferred to and work at another. This became known as *vendor lock-in* , a strategy often used by vendors to ensure customer loyalty [Sat+13; SRC13; OST14]. Some open standards were created to remedy this situation, such as the Open Virtualization Format (OVF) developed by the Distributed Management Task Force, but did not secure widespread adoption by big cloud Infrastructure Providers. A particular challenge to researchers and builders of open source cloud management software was therefore to make their software suites truly compatible. **Paper I** sought to address this need by providing an API that would allow cloud management software suites at different cloud Infrastructure Providers to efficiently migrate VMs between providers, possibly via a delegated chain of command spanning non-overlapping sets of collaborating Infrastructure Providers.

Vendor lock-in

The approaches taken in early international research projects such as RESERVOIR and subsequent EU-funded projects such as OPTIMIS [Fer+12] and VISION Cloud [Gog+12; Gog+13] were based on the assumption that the number of cloud Infrastructure Providers would be low and that federations could be managed by signing specific *framework agreements* across cloud sites. Such agreements stipulate the amount of resources contributed by each individual site and at what price. This was very much inspired by academic collaborations between supercomputing centers at a few universities and industry collaborations and partnerships, which typically involved only a few parties that had been awarded a grant to collaborate on a research project. A project on a completely different scale is the Worldwide Large Hadron Collider Computing Grid [Bir+14] (WLCG), which exemplifies how a massive international inter-organizational collaboration with a federated Grid can be successfully deployed and used for several years. The WLCG crosses organizational boundaries and the parties in the federation collaborate according to their framework agreements.

Framework agreement

3.2 Current Implementation of Cloud Federations

A new research frontier was established around the year 2015, focusing on a new generation of edge and "fog" cloud computing infrastructures[2] that are intrinsically reliant on inter-site collaboration. Several authors have argued that while general cloud computing does not *require* federations, edge computing does [YLL15; Lee16; CZS17; AZH18; PMA17; Osa+17; Mou+18; PDT18; Kha+19; You+19].

We argue that the benefits of cloud federations have not been overlooked by large cloud Infrastructure Providers. However, their implementations of federations do not cross organizational boundaries at all (contrary to early idyllic academic visions [BYV08; SB10]). Rather, cloud Infrastructure Providers structure their cloud data centers as independent and autonomous cloud sites internally, to the benefit of both themselves and their customers.

On a smaller scale, i.e. within a single cloud site belonging to a major public cloud Infrastructure Provider, there are often multiple redundant data centers with independent power supplies, storage arrays, and network connections to the outside world. These are typically called *availability zones* or something similar. They operate as autonomous and independent entities, so a site with multiple such zones satisfies our definition of a cloud federation (Definition 2). The availability zones belong to what is commonly called a *cloud region*, and are thus designed to serve end users in geographic proximity to the cloud site.

Availability zone

On a larger scale, i.e. between cloud regions, collaboration over greater geographic distances is common in practice *provided that* the cloud regions belong to a single cloud Infrastructure Provider. However, interoperability on a technical level between different cloud Infrastructure Providers is rare outside of academia, because vendor lock-in is the norm for business purposes [SRC13]. Even within the context of a single cloud Infrastructure Provider, cross-region functionality is typically

[2]The term fog computing was intended to be reminiscent of a dispersed cloud with clear proximity to end users, and encompassed the entire spectrum of computation from centralized clouds to edge data centers and end-user (Internet of Things) devices. This thesis does not stretch as far as the fog around end users, so we will focus our discussion on edge computing.

limited to ease of integration, e.g., via single-sign-on user contexts and permission models that give access to equivalent resource types in different regions. However, cloud regions typically share very little data, and assets such as virtual machine hard drive images must be uploaded separately to individual regions with region-specific identifiers, even if their content is identical on a byte level. Thus, Service Providers are typically acutely aware that cloud regions are isolated from each other, and have to modify their deployment strategies accordingly.

Meanwhile, there is clear evidence that Service Providers are increasingly making use of multiple Infrastructure Providers in a multi-cloud strategy, and with a substantially lower amount of friction than in the early days of the cloud. This trend follows the same pattern as the other evolutions we have discussed: an enabling technology emerges, and co-evolution occurs between the cloud and the software used to operate and utilize it. In the case of Service Providers making use of disparate Infrastructure Providers, that enabling technology is containerized software and Kubernetes. Kubernetes was developed by Google to dethrone Amazon Web Services as the major and practically unchallenged public cloud Infrastructure Provider. Drawing on lessons learned from its internal container orchestration system Borg [Ver+15], Google released Kubernetes as an open source product to level the technological playing field and reduce barriers between cloud Infrastructure Providers. By raising the level of abstraction such that Service Providers did not have to target specific Infrastructure Providers anymore, it became perfectly reasonable to split software deployments across multiple cloud Infrastructure Providers and thereby gain greater redundancy and a broader global presence. Because Google was the underdog in the public cloud Infrastructure Provider market at the time, this move made perfect sense: Service Providers could, by basing their software delivery and management around Kubernetes-specific abstractions rather than concrete cloud-specific concepts, ensure that their cloud applications were more portable than ever before. However, this came at the cost of increased complexity: now Service Providers had to essentially manage a large number of Kubernetes-managed clusters at various Infrastructure Provider cloud sites instead of just interacting with a single Infrastructure Provider. In contrast to a true cloud federation, this is a noticeable

Kubernetes

backwards step and a shift of responsibility to Service Providers that smaller organizations may struggle with.

For researchers in the field, relying on Kubernetes to abstract away underlying technological differences between Infrastructure Providers meant that the problems of incompatible cloud sites could be relegated to the past. However, new problems arose. For example, how does one manage a federation of Kubernetes clusters that may comprise hundreds or thousands of edge sites? This question is addressed from a technical perspective in **Paper VI** and in works such as [Kim+19]. Another important problem is how to choose which sites to use when, and to determine what benefits may be obtained. A new range of cloud brokerage studies can help address these needs.

3.3 Future Vision of Cloud Federations

What will cloud federations of the future look like and what needs will they address? While the future is by definition unknown, history teaches us a few important lessons:

- Vendor lock-in is not going away. The major cloud Infrastructure Providers are rightfully hesitant to deliberately destroy the forced loyalty won by locking Service Providers into their platforms. Google's strategic open source play with Kubernetes to dethrone Amazon Web Services has been further developed into a proprietary service offering called Google Anthos[3], which promises to leverage Kubernetes and other technologies to help enterprises bridge the gap between private, edge, and public clouds. So while other infrastructure may be used, Google will still get paid for the underlying enabling technology, to which Service Providers will be locked in.

- Service Providers care more about cross-provider federation than Infrastructure Providers do. In part due to the point above about vendor lock-in not going away, and because Service Providers

[3]`https://cloud.google.com/anthos`

stand to gain more from using multiple and competing Infrastructure Providers, more real and practically implementable innovation is happening in this space.

- The scale of federations is increasing dramatically. With some industry leaders stating that practically every mobile base station or factory in a fully Industry 4.0-based scenario may be an edge site, we know that future federations will have to successfully incorporate hundreds or thousands of sites.

- Resources will be heterogeneous among sites and scarce within sites. Service Providers have become accustomed to cloud sites (i.e. availability zones or regions) with large amounts of available resources that can be treated as though they originate from a more-or-less homogeneous pool. Future-generation federated cloud and edge infrastructures will exhibit greater heterogeneity on a technical level (e.g., they may use different CPU technologies), and by definition, edge sites have far fewer available resources than large centralized cloud data centers.

Based on these lessons, we predict that future-generation federations will be (a) driven primarily by the needs of Service Providers; and (b) far too complex for Service Providers to manage. Therefore, software or brokerage services will be created to optimize application deployment across multiple Infrastructure Providers, taking heterogeneity and differences in pricing into account.

Perhaps the predominant "Infrastructure Provider" of choice in the future will actually be a broker that enables smart access to disparate resources across several Infrastructure Providers, rather than an actual Infrastructure Provider with physical hardware to manage[4]? There is thus a clear need for research on optimizing the use of disparate pools of available resources (Scheduling, Chapter 4), and on making the best possible use of the resources currently available to Service Providers (Quality elasticity, Chapter 5).

[4]After all, as Tom Goodwin noted [Goo15]: "Uber, the world's largest taxi company, owns no vehicles. Facebook, the world's most popular media owner, creates no content. Alibaba, the most valuable retailer, has no inventory. And Airbnb, the world's largest accommodation provider, owns no real estate." Why would cloud infrastructure offerings be any different?

Chapter 4

Scheduling

In cloud computing, *scheduling* is the process of determining how and when to allocate resources from available pools, and from which pools resources should be allocated, in order to meet an application's needs. These pools may be physical hardware in a data center (e.g., physical machines to host virtual ones), resources in a (virtual) machine to host a containerized application (e.g., scheduling within a Kubernetes node or cluster thereof), or remote cloud sites in a federation for hosting applications that cannot be served using locally available resources. The output is some kind of mapping between the entity requiring resources (application) and the pool of available resources, showing when and what resources will be allocated. Because resource availability is volatile, scheduling is an iterative process. To the extent that the underlying technology permits, scheduling decisions can be remade to make better use of resource pools. **Scheduling**

This chapter discusses scheduling as it relates to the overarching topic of this thesis, i.e. managing cloud resource scarcity. By way of introduction, we first note that scheduling is an optimization problem for which we must often make do with approximate solutions (Section 4.1). Next, we see how scheduling in cloud environments has co-evolved with Service Provider expectations (Section 4.2). In particular, commitments have become shorter, giving Infrastructure Providers more opportunities to optimize the use of their data centers via smart scheduling. Because local resources may be insufficient to meet demand, scheduling may also have to include other Infrastructure Providers in a federation (Section 4.3). Based on experience and current research directions, we conclude the chapter with a look toward the future of scheduling in cloud and edge computing (Section 4.4).

4.1 Scheduling as an Optimization Problem

On an abstract level, a data center can be viewed as a set of physical servers that each have different amounts of spare capacity, and application instances can be seen as entities requiring capacity. In cloud computing, schedulers are used to produce both task *schedules* and service *placements*. Both are assignments of application instances to (physical) servers, with the difference being that the runtime durations of services are unknown when scheduling. Viewed this way, scheduling is an optimization problem, i.e. one in which the goal is to find the "best" schedule/solution among many possible ones. But what is "best" in this situation?

Schedule and Placement

Optimization problem

Optimization problems are well-studied in mathematics, economics, and computer science, and can be expressed using formal models that leave no room for ambiguity or misinterpretation. The general process of optimization involves finding inputs $x \in \mathbb{R}$ that minimize an *objective function* $f(x) \in \mathbb{R}^n \to \mathbb{R}$ for the given inputs x, subject to a set of *constraints* $g(x)$ that also depend on x (constraints are either inequality constraints or equality constraints, $\leq$ or $= 0$). These parts play together nicely and let us express a multitude of problems in a generic way so that *solvers* can find the best possible solution given our problem description.

For instance, when scheduling application instances to physical machines, we can express the maximum amount of available resources for each physical machine (constraint), require that an application instance be scheduled to at most one physical machine (constraint) and that all application instances we know of must be scheduled somewhere (constraint), and state that solutions should be ranked based on the number of powered-on servers, with solutions using fewer servers having higher ranks (objective function).

Because virtual machines are typically not divided into fractions and deployed across multiple physical machines, we can often express the scheduling optimization problem as one that requires integer solutions, which may be obtained using *integer linear programming* (ILP). **Paper III** uses this formulation, as does the work of van den Bossche et al., which serves as an approachable introduction to the topic [VVB10]. In ILP formulations, singular resources such as processing time on a CPU core can be shared by dividing them into smaller units and assigning integer quantities of these smaller units to the different application instances. Kubernetes uses this approach: containers are assigned a certain number of millicores (thousandths of a CPU core). Thus, a container could be assigned 500 millicores, leaving the other half of the CPU time available for some other container.

Integer linear programming

Much of the published work on cloud scheduling and data center optimization uses an optimization problem formulation. These works differ with respect to the components they include: they may employ different constraints to model different aspects of the problem domain or different objective functions that prioritize, e.g. consolidating application instances onto fewer physical machines (to reduce power consumption and operational expenditures), spreading application instances out (to reduce performance interference and increase resilience to individual physical server failures), or maximizing overall performance and data center utilization.

Unfortunately, many scheduling optimization problem formulations are very complex (in the NP complexity class [Ull75]), so solving them is very computationally expensive. Therefore, results are typically not guaranteed to be optimal but rather approximations informed by *heuristics* (computationally cheap shortcuts leading to inexact but reasonable solutions), because truly exhaustive testing of all possible solutions would be infeasible and highly questionable from a cost-benefit analysis perspective: the cost in terms of time and resources of finding more optimal solutions will outweigh that of using a sub-optimal solution. Therefore, in practice, even approximate optimality can be dismissed to reduce computational costs, and scheduling decisions are often based on the results of rather simple greedy methods instead. Solutions must still take constraints into account, and some effort is typically made to choose a "good" (if not "best") solution from a range of possibilities.

Heuristics

Scheduling can be performed either by a central scheduling component, or in a distributed manner [TW84], e.g., by using auctioning protocols [Wel+01; Att+06] in which entities (cloud or edge sites, or physical machines within a site) "compete" with each other by bidding on application instance allocations. Decentralizing scheduling removes bottlenecks and the single point of failure inherent in centralized systems, and can be made quite efficient [Hua+13]. **Paper VI** proposes a collaborative decentralized scheduling method for use in Kubernetes federations.

Decentralized scheduling

4.2 Co-development of Scheduling and Cloud Computing

A recurring theme of this kappa is that user expectations define what technical solutions are possible, and technological advances can only be made via circular co-evolution with changes in user expectations. Scheduling is no different. In the early days of cloud computing, most users expected forklifted virtual machines to replicate the properties of machines in private data centers, so Infrastructure Providers offered guarantees of always-on services with no downtime and no disturbance from other cloud customers. Schedulers thus had to maintain the *illusion* that the Service Provider's virtual machines had dedicated resources at their disposal.

At best, Infrastructure Providers could *in theory* use virtual machine migration to make new scheduling decisions, but even the largest public Infrastructure Provider, Amazon Web Services, seemingly does not do it[1]. Although great advances have been made in techniques for performing seamless live migration across physical machines [Svä+11; Voo+09; Xu+14; BKR10], their performance costs are apparently deemed too great in practice.

However, offering dedicated resources represents a great missed opportunity cost to Infrastructure Providers[2]. There has been extensive

[1]As of June in 2020, Amazon Web Services customers will still receive notification emails when the physical machine upon which their virtual machines are deployed is due to be retired, stating that this will cause the forced termination of the virtual machine. This implies that little has changed regarding the original decision to not migrate virtual machines, even if customers are inconvenienced as a result.

[2]As of June 2020, the pricing scheme of the European cloud Infrastructure Provider Scaleway shows that roughly equivalent dedicated bare metal servers cost about

research on how to safely perform *over-booking*, i.e. selling the same resources to multiple parties with the statistically backed hope that they will not all *actually* require all of the sold resources at the same time (with the provider paying a penalty on the off chance that this occurs) [BE11; Váz+13; TT13; TT14; TLE16]. This practice is commonly used in other fields - for example, airlines routinely over-book flights in the knowledge that no-show travelers are common enough to safely sell a number of additional seats per flight with the calculated risk of having to ask travelers to take a later flight if seat availability is actually exhausted.

Over-booking

Seeking larger profit margins, cloud Infrastructure Providers saw an opportunity to offer a tiered pricing model in the hope that a market to exploit it would appear. This gave rise to virtual machines with *burstable* performance profiles, where the machine accumulates performance tokens at regular intervals (up to some limit), and will dynamically use them to operate at a higher than normal rate of performance. Once the performance token pool is empty, the virtual machine operates at its normal comparatively slow speed. This was an attractive option for users who knew that their servers were only used to the 15–20% that research has shown is common [Vog08]. The reasoning was that if the server will only need its top performance mode a fraction of the time, why pay for maximum capacity *all* the time? Because virtual machines with burstable performance profiles had different demands and user expectations to regular virtual machines, they allowed schedulers to make safer and more extensive over-bookings and thereby increase profits.

Burstable performance

Similarly, cloud Infrastructure Providers created *spot instances*, which offered no uptime guarantee at all but were significantly cheaper than other alternatives. These allowed schedulers to apply *backfilling*, a technique used extensively in parallel and high-performance computing [MF01; Sri+02; TEF07] whereby gaps in schedules are filled with small and preemptible workloads. This essentially allowed Infrastructure Providers to sell spare capacity that would otherwise have been wasted at a low price and with clearly stated limitations. Thus, user expectations were adjusted and technology could be developed to exploit these new capabilities. The adoption of batch systems with frequent

Backfilling

50% per month than virtual servers, which hints at the increased margins efficient scheduling can offer in this space.

result checkpointing and systems focused around work queues enabled Service Providers to do processing inexpensively. Economic models have since been developed to clarify how spot instances can be exploited most effectively in specific use cases [Amb+19; Ali+19; SAS19; Irw+19].

While many cloud workloads come and go, others are very long-lived. Thus, scheduling decisions that are optimal at one point in time may leave a physical machine powered on for a long time afterwards to serve a single long-lived virtual machine. For example, company e-mail servers can potentially have years of uptime, whereas a worker node in a batch processing system may only exist for a few hours. Unfortunately, when provisioning a virtual machine, an Infrastructure Provider cannot know how the Service Provider intends to use it or what its likely lifetime will be. This is a problem for cloud scheduling, and leads to poor resource utilization for Infrastructure Providers.

As cloud-native technologies matured and software architecture developed to more clearly separate stateless and stateful components, virtual machines became more easily replaceable and short-lived. Thus, scheduling commitments could also be shortened, allowing more optimal resource usage. Additionally, spot instances enabled gap-filling in schedules, and because the expectations of Service Providers had been properly set by Infrastructure Providers, all parties were aware that spot instances were ephemeral.

However, starting up a virtual machine is still a rather slow process[3], so there has been a move to enable better scheduling and faster implementation of new decisions by relying on significantly smaller execution units such as containers and functions (see Section 2.3).

By clearly stating what is being offered and at what price, Infrastructure Providers can ensure that the expectations of Service Providers are set accordingly. Moreover, technology has evolved to take advantage of these offerings while avoiding their most serious pitfalls. By having Service Providers clearly show which workloads may be long-lived and which ones will certainly not be, Infrastructure Providers can mitigate the ill effects of suboptimal long-term scheduling decisions by com-

[3]As of June 2020, the AWS FAQ for the EC2 service still states that it can take up to 10 minutes to start a virtual machine.

pensating and allocating many short workloads to achieve high overall utilization and profitability.

4.3 Scheduling Across Cloud Federations

Perhaps the greatest selling point of cloud federations is the ability for an Infrastructure Provider to meet current resource demand using resources at another cloud site. Schedulers in cloud federations must therefore at minimum be aware of the cost of using resources from other sites in the federation. These costs may be the same as those paid by any other customer or reduced by agreed-upon framework agreements. Including such federated resources in an optimization problem formulation is merely a matter of using an objective function that assigns different costs to local deployment (based on factors such as the power and maintenance costs of the physical machines) and remote deployment, with the latter being assumed to have a very large capacity. In practice, two of the guiding principles of this thesis, independence and autonomy, dictate that Infrastructure Providers cannot let each other know their true resource availability at any given moment (competing businesses would never disclose this information). This issue aside, there is another important question to consider: assuming that remote deployment is possible, can any virtual machine really be placed elsewhere in a federation?

The answer is, of course, no. Some groups of virtual machines must be deployed in close proximity to one-another to maintain acceptable performance. In other cases, legal requirements may dictate that data remain in a certain geographical area. Unfortunately, virtual machines are essentially black boxes from the point of view of the Infrastructure Provider. What then can be done to avoid making mistakes that might breach these requirements?

Paper II and **Paper III** propose a solution to this problem, namely to let the Service Provider explicitly express the "structure" of the deployed application, along with *placement constraints*. These placement constraints specify which components must be co-deployed (e.g., components that need to communicate frequently), which ones should never be co-deployed (e.g., replicas), and possible geographical restric-

Placement constraints

tions. Similar approaches have been suggested by other authors such as Kim et al. [Kim+19].

Figure 4.1 exemplifies such placement constraints. The service depicted in the figure is a typical three-tier web application, i.e. one in which the front end web server, business logic processor, and database service are separated each other. The database service is replicated such that there is one primary database instance and multiple secondary instances for redundancy purposes. The service structure is hierarchical, and a top-level placement constraint states that all virtual machines (instances) in the service have an affinity for Europe. They can thus be placed anywhere in a federation, as long as the cloud site is in Europe. An additional constraint stipulates that an internal network must be available to all instances; therefore, if multiple cloud sites are used, the cloud Infrastructure Provider must extend the internal network by creating Virtual Private Networks (or similar) across cloud sites.

The Primary DB instance has an anti-affinity placement constraint on the host (physical machine) level toward all Secondary DB instances. Similarly, all Secondary DB instances have anti-affinity constraints against each other on the host level. These placement constraints ensure redundancy: if a physical server should halt and catch fire, it can at worst take out *one* instance of the database service.

The placement constraints suggested in **Paper II** and **Paper III** thus grant the Service Provider some influence over how the service is placed, but not *control* over it. The Service Provider cannot demand that a particular physical machine be used, as that would violate the autonomy of the Infrastructure Provider.

4.4 Scheduling in Future Cloud and Edge Computing

We believe that federations represent the future of cloud and edge computing. This implies that cross-federation scheduling will be needed to fully exploit resources at multiple sites, and to make smart choices on behalf of users by taking costs into account and ensuring that geographical distances are kept reasonable given the structure of the deployed application. We also believe that for the foreseeable future, Service

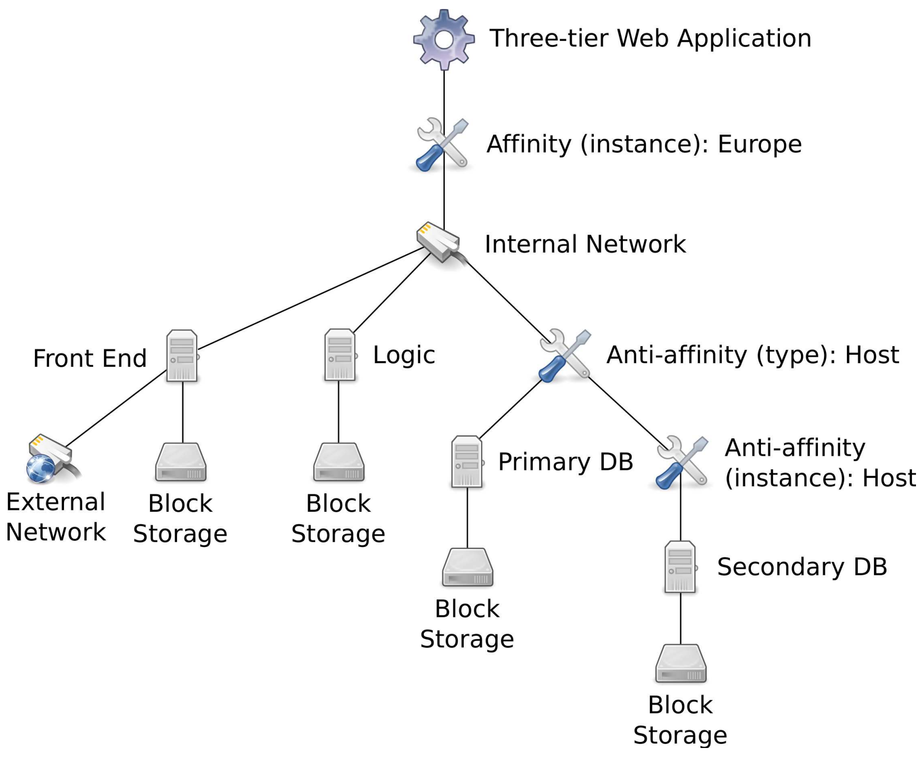

Figure 4.1: Placement constraints of the type proposed in **Paper II** and **Paper III** for a three-tier web application with a replicated database service. The replicated database service ensures redundancy using anti-affinity placement constraints.

Providers will have to own and refine the solutions that enable federation (see 3.3). Infrastructure Providers cannot be relied upon to do this because they are not generally interested in making it easier for their customers to do business with their competitors.

Illustrating the gradual movement towards scheduling tailored to federations, Kubernetes now includes native support for inter-Pod affinity and anti-affinity placement constraints similar to those proposed in **Paper II** and **Paper III**. Additionally, the Kubernetes Special Interest Group is developing the kubefed federation controller to allow scheduling across a federation to be guided by tags that could express geographical locations. The designers of kubefed (who work for major

Infrastructure Providers) have stated that the software is intended to handle federations consisting of dozens of sites. However, edge computing may involve the use of federations that are multiple orders of magnitude larger. **Paper V** therefore proposes a more scalable approach to scheduling than that adopted in kubefed. By ensuring compatibility with kubefed, our approach retains the ability to meet user expectations relating to controls such as geographical location restrictions. We believe that we will see similar user-driven endeavors to make unfettered use of technology, rather than accepting the restrictions imposed by competing Infrastructure Providers.

Perhaps the future of scheduling will include dedicated businesses whose operations revolve solely around offering smart federation and brokerage services that draw on advances in research to solve the optimization problem so that Service Providers themselves do not have to.

Chapter 5

Quality Elasticity

With a finite amount of resources and time, only a finite amount of work can be done. In this chapter, we consider time limitations that may be imposed by users who will lost interest if forced to wait for too long [Pog+14] (web users are typically willing to wait no more than 2 seconds [Nah03]) or by technology, such as network connection timeouts. As shown later in this thesis, resource limitations will always exist, whether due to the Infrastructure Provider's bounded capacity or the Service Provider's bounded budget.

This chapter makes the case that a growing number of researchers and industry practitioners consider it better to do *something* (deliver some service to some users) when facing resource scarcity rather than to do *nothing* (provide no service to any users) because of problems such as timeouts. This can be done by modifying applications to work with momentarily imperfect data, which increases their scalability and agility, allowing them to better cope with resource scarcity. The sections below present a brief introduction to approximate computing (Section 5.1) and discuss examples of its practical use in databases that reduce quality by relaxing guarantees and offering so-called eventual consistency (Section 5.2) as well as in certain applications (Section 5.3).

Most modern software does not adapt itself to current operating conditions and therefore becomes saturated in times of resource scarcity [Kle+14a], i.e. unable to handle the current load with the available resources. This results in rapidly increasing response times or, worse, timeouts caused by failure to respond quickly enough. This is unsatisfactory to both Service Providers and end users. However, a growing body of research shows that this mode of operation can be avoided.

Suppose that software instead adapted to current resource availability, for example by using a more computationally expensive algorithm when CPU resources are abundant and a computationally cheaper one when such resources are scarce. **Paper IV** introduces the term "quality elasticity" to describe behavior of this sort; a modified definition of this term is given below:

Definition 3 (Quality elasticity) *The ability of software to adapt both its mode of operation and its result output quality in response to current operating conditions, achieved by reacting to both its (a) statically determined* runtime environment configuration *and (b) dynamically determined* current resource availability.

In this definition, the term *mode of operation* encompasses various methods discussed in this chapter, including relaxing consistency requirements for databases [Chi+12]. Adapting *result output quality* entails adjusting the quality of the software's output by, e.g., serving a possibly stale cached response instead of generating a fresh one, or by performing simpler database queries that generate mostly correct results.

The *runtime environment configuration* is the amount of memory and/or CPU allocated to the VM or container in which the software is running. Information on the runtime environment could (for example) cause the software to choose a more memory-intensive algorithm rather than a CPU-intensive alternative (i.e. one that caches partial results in preference to recalculating them) if the memory allocation is greater than the CPU allocation. Finally, (dynamic) *current resource availability* is the instantaneous resource availability, determined from the most recent possible readings. While dynamic resource availability readings are much more informative than static ones, and are therefore preferred, observing systems during runtime incurs a performance penalty [CP17], necessitating a trade-off.

5.1 Approximate Computing Primer

As noted in Chapter 4, some problems are too hard to solve fully within a reasonable time frame or with reasonable resource expenditure. In such situations, solutions that are "good enough" (*satisficing* solutions according to the terminology of Simon [Sim56]) may be deemed acceptable. While it can be argued that any floating point operation in a computer is rounded off and therefore all such results are numerical approximations [XMK16], *approximate computing* as a field of study came into its own in the 2000s [XMK16; Mit16; HO13]. Many highly cited works on approximate computing focused on increasing energy efficiency by minimizing the amount of computation required to get "good enough" results.

Approximate computing

This required application components to be designed from the start to both accept approximate results from other components as input, and to themselves be able to produce approximate results. If applications are allowed to return inexact results to improve energy efficiency or (drastically) reduce processing times, the quality of the results must also be monitored to assess the degree of performance degradation [Yaz+17], and there must be a way to decide *when* it is appropriate to degrade quality because some parts of an application will be more resilient to approximation than others [Chi+13]. For instance, it may be appropriate to encode video at a lower quality (bit rate) if the results will hardly be noticeable and doing so would conserve resources. Frameworks have been developed to let Service Providers apply the principles of approximate computing at scale [Goi+15; Quo+18].

5.2 Eventual Consistency in Databases

The adage in computer science that "premature optimization is the root of all evil" (attributed to Sir Tony Hoare and popularized by Donald Knuth) helps us identify the critical paths of our applications [CSK02], that is to say, the performance bottlenecks through which most key operations must pass. For many applications, this bottleneck is the database, which is used to store and retrieve results generated during the application's use. Databases thus contain the "truth" of an application. Can such databases be made approximate?

Before directly addressing that question, let us first note that databases are complex systems that essentially exist to enforce rules about the data they store. Without guarantees that application developers can easily understand (e.g., guarantees about what will happen if there are conflicting value updates), dependable applications would be exceedingly difficult to produce. Common database software such as PostgreSQL and MariaDB/MySQL makes very strong guarantees about how operations against data are performed:

- **Atomicity**: operations against the data either happen in their entirety or not at all;
- **Consistency**: operations on data cannot cause the database itself to enter an erroneous state;
- **Isolation**: updates from several sets of operations do not interfere with each other; and
- **Durability**: once a database commits to a set of updates, the new state cannot be lost, even due to power failure or similar.

ACID

These guarantees are referred to as the ACID properties [HR83]. Considerable computational work and bookkeeping is performed to ensure these ACID properties are offered by database software, because databases offer concurrent access to the stored data.

To make matters more complex, an application may outgrow the performance or availability offered by a single database server, at which point the database must become a replicated or otherwise distributed service. When that happens, software designers have two choices. One is to require their replicated database service to spend a lot of additional computational effort to maintain the ACID properties and thereby essentially hide the fact that the service is distributed across several nodes. In this case it would, for instance, be forbidden to seemingly "forget" a value update. Instead, once data has been written to the replicated database service, any and all upcoming queries must immediately thereafter get answers with the same data.

In a highly influential keynote presentation in 2000, Eric A. Brewer argued that any shared-data system can achieve **at most two** of three key properties – Consistency, Availability, and tolerance to network Partitions [Bre00]. This became known as the CAP theorem. Intuitively, the CAP theorem makes it impossible to design a shared-data system (i.e., a replicated or otherwise distributed database) that simultaneously ensures that all data is consistent at all times, data is always available, and network outages (partitions) between replicas in the system can be tolerated. The theorem has since been criticized as being insufficiently nuanced [Aba12], and it has been suggested that network partitions may not be as problematic as was originally claimed [Bre12b]. A later refinement by Abadi et al. [Aba12] states that during normal operation (i.e. without network partitions), distributed database systems typically have to favor either latency or consistency. However, the core message remains valid: when harnessing the collaborative power of distributed systems, some concessions must be made.

CAP theorem

For instance, software designers may construct their systems such that momentary inconsistencies can be accepted in the knowledge that the system will have an *eventually consistent* view of the data. Updates may thus not be reflected immediately across the entire database service, but a behind-the-scenes state reconciliation process will ensure that members of the service eventually agree on which values are the current ones [Bur14; BG13; SK17; GA02]. Databases can achieve this by dropping the ACID properties entirely, or by instead offering the so-called BASE guarantees:

Eventual consistency

BASE

- **Basically Available** - applying Brewer's CAP theorem [Bre00], the **consistency** property is relaxed to favor availability such that the replicated database is always available to serve read and write requests, even in the event of network partitions;

- **Soft state** - the state of underlying data can change without further input (enabling reconciliation after operations have been serviced); and

- **Eventually consistent** - the state of the system will eventually be consistent, given enough time.

BASE systems drop expensive bookkeeping processes and instead favor doing at least *something* right now, rather than spending resources to satisfy the strict immediate correctness requirements of ACID. This provides an immense performance and scalability boost [Bur14], but because it sacrifices consistency (i.e. at least momentary "correctness") for performance, software designers must be aware of its pitfalls [Llo+14; BD13; BT11]. Eventual consistency can therefore be viewed as something akin to approximate computing in the context of databases. Software developers became familiar with databases of this sort largely due to the immense popularity of MongoDB and the "NoSQL" movement in the early 2010s.

Of particular interest to us in this thesis is a particular kind of eventually consistent database, one that is designed from the start to avoid the expensive bookkeeping needed to keep multiple database replicas in sync, treating partitioning (and thus loss of availability) as a natural consequence of distributed systems [Bre12a]. **Paper VI** uses the so-called Conflict-Free Replicated Data Types (CRDTs) introduced by Shapiro et al. [Sha+11b; Sha+11a] in 2011 to implement a distributed database underpinning a federation control plane for edge-scale Kubernetes federations.

CRDT

Conflicts occur in distributed databases when members of the distributed system do not agree on what values are correct. Conflicts can either be pessimistically assumed to be frequent and avoided by using protocols such as multi-phase commit protocols, or optimistically assumed to be rare and resolved when detected. CRDT-based databases, and particularly those using *convergent replicated data types* (CvRDTs), take a highly optimistic but non-naïve approach: by only working with data types that exhibit mathematical properties such as commutativity or monotonicity in a semi-lattice[1],they make conflicts provably impossible [Sha+11b]. Members of a such a distributed database will maintain their own local state and distribute it to all other members indefinitely. Value updates will be made to the local state and merged with the data received from other members such that all members eventually receive

[1]A semi-lattice is an algebraic structure $\langle S, \cdot \rangle$, where $\cdot$ is called the semi-lattice operation and is an infix binary operation and where the following identities hold: associativity, i.e. $x \cdot (y \cdot z) = (x \cdot y) \cdot z$, commutativity, i.e. $x \cdot y = y \cdot x$, and idempotency, i.e. $x \cdot x = x$. Importantly in the context of distributed systems, these properties mean that the order of operations does not matter and that operations can be repeated with idempotency, that is, without affecting the result "again" if repeated.

and merge the same updated state. The commutativity of operations ensures that local states will end up being consistent.

Alternatively, one could have a database using *commutative replicated data types* (CmRDTs), in which operations rather than internal states are transmitted to other members of the distributed database. These require that operations be commutative but not necessarily be idempotent, which means that the underlying data distribution protocol must ensure that operations can neither be lost nor repeated. As such, they require more complex data distribution protocols than CvRDT-based systems, but the messages passed between members are smaller. The use of less complex data distribution protocols is useful when poorly inter-connected large-scale systems that span across large geographical distances must be supported, as in edge computing. We therefore focus the remaining discussion on database systems using state-based CvRDTs rather than operation-based CmRDTs.

Some examples may be in order. Imagine a Boolean variable with the initial value "false" along with a rule saying that if it ever gets set to "true", it must stay that way. Whenever a member of a distributed system sees such an update from a client, no other member can refute the value update by saying that the value should be "false". Similarly, setting it to "false" again has no effect — the value update is idempotent. Next, imagine a vector of values with as many items in it as the number of members in the distributed system, and an indexing rule that ties an item in the vector to a particular member. Now add the rule that each member can modify only "its own" value, but read all others. Members can then modify the sum of the vector by modifying their own value slots without risk of conflicting value updates.

CvRDT-based databases typically use an eventually consistent data distribution protocol to spread local states between members. In one such family of protocols known as Gossip [Dem+87; JHB01; Bir07], data is permeated throughout the system using a method akin to gossiping among social peers. Parties acquire data updates passively by continuously sending and receiving messages bearing the latest data they are aware of, and by requesting updates specifically if it becomes apparent that they may have fallen out of the loop and missed something.

Gossip

The massive scalability of CRDTs has been proven in practice [Yu12; Lv+17], and while the underlying data types are chosen specifically for compatibility with the mathematical properties required by the databases, general-purpose data structures can be implemented on top of them [Bie+12], making CRDTs suitable for general-purpose databases such as the "Riak KV" key-value store.

Paper VI argues that a distributed CRDT-based database is a suitable eventually consistent data storage service on which to build a federation control plane for edge-scale Kubernetes federations. Eventual consistency matches the expectations that Service Providers already have of Kubernetes, but the wide-area network performance and scalability of CRDT-based databases far surpasses that of the etcd database that Kubernetes uses on a per-cluster level.

5.3 Adjusting Application Output Quality to Resource Availability

According to Definition 3, software that can adapt its output quality to resource availability is quality elastic. The literature offers many examples of reducing output quality based on resource *scarcity*, but fewer of increasing quality when resources are abundant. This may be because reducing output quality is seen as an action taken to handle an emergency rather than a tool for application performance management.

A key property of quality elastic software is the ability to choose between different code paths that offer similar functionality but have different implementations and resource requirements, such as those presented in [DR97] and [Ans+09]. The use of computationally less expensive code paths when resources are scarce has been successfully applied in web server systems for both static [AB99] and dynamic [Phi+10] content. Software with this capability has been described as *self-adaptive* [Che+09; Kep05; KM07]. The adoption of self-adaptive software has not solely been motivated by a desire to adapt to changes in resource availability; it has also been used for end user classification [Mer+11] and to automatically determine usage patterns and requirements [Chi+12].

Self-adaptive software

The work presented in **Paper IV** was strongly influenced by Brownout [Kle+14a], which applied a conceptually very simple framework for serving optional content and a more involved control theoretic approach to determine when to serve with or without said optional content. The motivating example of Brownout was an e-commerce site in which product recommendations were considered optional content when viewing a product detail page: showing recommendations benefits the Service Provider because they increase sales [FH07; FH09], but they are not strictly necessary from the end user's perspective. In the original Brownout paper, a control theoretic approach was used to decide to either generate or not generate responses with optional content based on estimated response times. It thus enabled quality elasticity by allowing for quality *reduction* if resource scarcity prevented requests from being processed fully. Many subsequent works have adopted goals and formulations similar to those of the original Brownout study [Kle+14b; XB19] and used them to optimize scheduling [Tom+14; Pap+17; XDB16; XTB19].

Brownout

It is also important to consider what to do if resources are abundant. In **Paper IV**, we argue that it would be useful to have additional more computationally expensive code paths to take when resource availability permits so as to *increase* output quality, rather than just providing it at the normal or reduced levels. We therefore modified the Brownout simulator to account for this possibility, and obtained promising results.

Caching

Unfortunately, retrofitting an existing application for quality elasticity is time consuming. Caching is mainly used as a way to improve application performance, and is not traditionally seen as a quality elasticity tool in itself. However, basing computations on possibly stale data constitutes a reduction in quality. In **Paper VII**, we quantify the effects of dynamically caching inter-service communication in a micro-service context. Rather than caching objects close to the end user, where caching times are limited by the most frequently updated data item, we instead employ fine-grained caching between application components. This makes it possible to serve fresher data to end users since even if only a part of the response is cached, inter-service requests can be significantly reduced by caching data that changes only rarely. Our caching infrastructure requires no application modification but reduced the number of inter-service queries by 80% in a real micro-service application.

Chapter 6

Summary of Contributions

This chapter summarizes the papers comprising this thesis and shows how they relate to the overall research goal and research objectives, and the guiding principles of the thesis. An overall outline is given first, after which the papers are presented in more detail in chronological order, with descriptions of the author's contributions.

6.1 Outline of Contributions

The research goal of this thesis is to explore methods for managing cloud resource scarcity. This was done from two perspectives: that of a Service Provider (**RO1**) and an Infrastructure Provider (**RO2**). The tested approaches can be considered to have their roots in the three pillars of this thesis: cloud infrastructure federations, scheduling, and quality-elasticity. Figure 6.1 shows which papers address the problem from which perspective and which approach was used. Note that **Paper V** lies outside this figure because it is not tied to any of the three pillars: it presents an experimental study on the adverse effects of resource scarcity on the control plane and deployed applications in a cloud environment rather than contributing solutions to these problems.

As Figure 6.1 shows, some of the papers span the divide between Service and Infrastructure Providers, and propose collaborative solutions. This approach is consistent with a central argument of this thesis, namely that the cloud and the applications deployed onto it have co-evolved, leading to the emergence of cloud-native software. **Paper I**

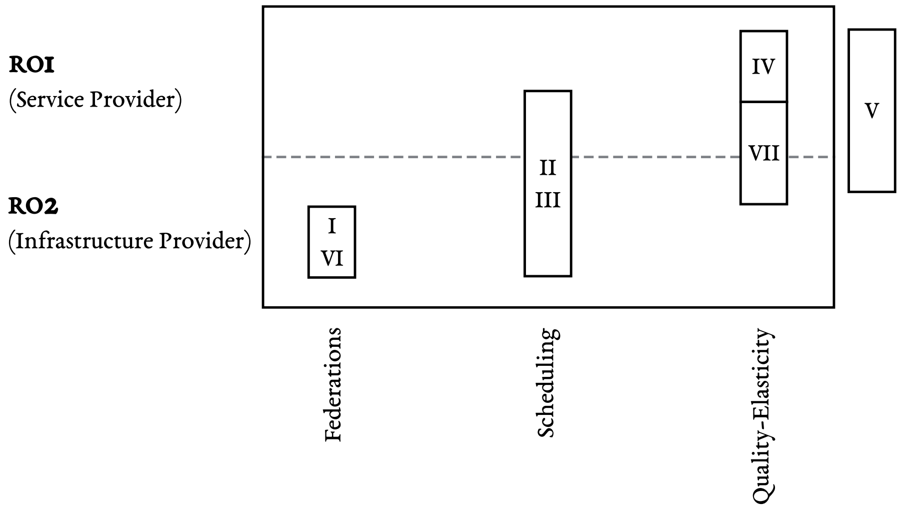

Figure 6.1: Mapping of the contents of the included papers to the research objectives of this thesis (Section 1.1) and its three pillars. Note that some papers span the Service/Infrastructure Provider divide because they explore collaborative solutions. **Paper V** is not tied to any particular pillar because it highlights challenges and lessons learned when performing experimental evaluations in complex cloud environments. However, its results strongly influenced the experimental strategies adopted in other papers.

outlines how Infrastructure Providers can extend their own capacity by renting that of others to create a collaborative cloud federation. This may be done to ensure that Service Providers do not perceive resource unavailability at their chosen cloud Infrastructure Provider. Unfortunately, Service Providers might object to such an approach, possibly because their cloud deployment has performance-sensitive components that must be deployed in close proximity to each other. Alternatively, there may be legal restrictions on where certain components can be placed. Service Providers therefore need a way to specify such requirements. **Paper II** and **Paper III** provide a language for this purpose (which constitutes the main focus of **Paper II**) and show that it is mathematically and practically feasible to take these additional constraints into account during scheduling (the main focus of **Paper III**).

Paper V experimentally explores and evaluates the inability of core components of Kubernetes, including its current scheduler, to function in situations when resources are scarce (or at least not fast enough). The results obtained show that even with careful preparation and a solid understanding of how to perform load experiments against an application, the underlying platform cannot be ignored. These findings will help to guide future research by highlighting pitfalls and facilitating their detection and avoidance. The experience gained during the work presented in this paper directly influenced the experimental approach applied in **Paper VII**, where algorithmic goodness was measured in terms of numbers of bytes sent across the network rather than by using raw performance metrics.

While the earlier papers focused on what Infrastructure Providers can do, the later papers focused on ways for Service Providers to manage cloud resource scarcity. Auto-scaling represents one possible solution (and is addressed in **Patent I**, which is not part of this thesis), but it is a coarse-grained approach that works on a timescale of several minutes whereas end-user satisfaction can change on a sub-second timescale. Therefore, faster responses to resource unavailability are needed. **Paper IV** suggests that this can be achieved via automatic and voluntary quality adjustments on the Service Provider's side: the quality of results can be increased when resources are plentiful and reduced when resources are scarce.

The results of **Paper IV** are interesting and part of a subfield that warrants more attention, but the approach it suggests places a heavy burden on cloud application developers because it requires them to implement complementary and quality-differentiating services. **Paper VII** therefore suggests an alternative approach based on automated response caching as a way to reduce the resource requirements of a cloud application deployment. The goal is to avoid any need to change the application itself by dynamically adjusting the amount of caching done within the system.

Finally, **Paper VI** (along with **Patent II**, which is not part of the thesis) revisits the problem domain of **Paper I**, but several years later and with a different goal: to decentralize the entire control plane between edge computing federations of massive scale with dynamic execution.

This is something that cannot possibly be achieved using approaches built for mere handfuls of cloud Infrastructure Providers.

The research presented in this thesis has thus explored how both cloud Infrastructure Providers and Service Providers can manage cloud research scarcity issues that arise when current resource requirements exceed resource availability. The later papers have gravitated toward solutions that can be implemented by Service Providers because the author's professional experience indicates that Service Providers are more strongly incentivised to address momentary cloud resource scarcity than Infrastructure Providers.

6.2 Paper I

E. Elmroth and L. Larsson, "Interfaces for Placement, Migration, and Monitoring of Virtual Machines in Federated Clouds", *Eighth International Conference on Grid and Cooperative Computing (GCC)*, Lanzhou, Gansu, 2009, pp. 253-260.

Paper I introduced the notion of an inter-cloud API that allowed one cloud to delegate VMs to another (**GP1**) and also enabled VMs be migrated directly in one step from the source cloud to the ultimate destination cloud in the event of multi-step delegation (**GP2**). The paper thus relates to **RO2** and the overall research goal in that a cloud Infrastructure Provider could, for instance, decide to delegate a set of VMs to a collaborating Infrastructure Provider to compensate for resource scarcity.

This work was conducted in the context of the RESERVOIR EU FP7 project on federated cloud infrastructures, and the work done within that project on inter-cloud interfaces inspired the Open Cloud Computing Interface (OCCI), a vendor-neutral cloud API [1].

Author contributions *Lars authored the paper in its entirety, with key invaluable feedback from Lars' then sole supervisor, Erik Elmroth. It should be noted that when this paper was written, our research group listed authors alphabetically, so the author list does not reflect the magnitude of each author's contribution.*

[1]`https://occi-wg.org/about/specification/` has links to all documents in the OCCI specification suite

6.3 Paper II

E. Elmroth and L. Larsson, "Scheduling and Monitoring of Internally Structured Services in Cloud federations", *2011 IEEE Symposium on Computers and Communications (ISCC)*, Kerkyra, 2011, pp. 173-178.

Paper II introduced a language that lets Service Providers tell Infrastructure Providers which components of their application are related to each other, and whether they have an affinity or anti-affinity toward each other. These (anti-)affinities constitute *placement constraints*. Placement constraints can affect service deployment on different levels, e.g., "this component must never be placed on the same host as another instance of the same component" (for redundancy), "this component must be placed in the same cloud as this other component" (for efficient networking), or "these components may never be placed outside of country X" (for legal reasons).

A crucial aspect of this contribution is that the Service Provider cannot dictate exactly *how* the cloud Infrastructure Provider(s) should place resources (**GP1**); instead, the Service Provider's requirements serve as inputs for the Infrastructure Provider's placement optimization algorithms. Thus, the Infrastructure Provider is still free to optimize within their data center. **Paper II** addresses objectives **RO1** and **RO2** because the Service Provider is granted influence, but not control, over the Infrastructure Provider.

Nothing comparable to the proposed placement hints was offered by cloud Infrastructure Providers when the paper was written, but similar functionality has since appeared, notably in the form of AWS Placement Groups[2] and Kubernetes Pod affinity specification[3].

Author contributions *Lars and Daniel Henriksson co-formulated the core idea of representing service structure in the manner described in the paper. The monitoring framework mentioned in the paper was Lars' idea entirely. Lars then authored the vast majority of the paper. Credit, as agreed between Daniel and Lars, should be split evenly, and joint first authorship*

[2]`https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/placement-groups.html`

[3]`https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/`

for the papers was implemented by assigning first author position to one for ***Paper II*** *and to the other for* ***Paper III****.*

6.4 Paper III

D. Espling, L. Larsson, W. Li, J. Tordsson, and E. Elmroth, "Modeling and Placement of Cloud Services with Internal Structure", in *IEEE Transactions on Cloud Computing*, vol. 4, no. 4, pp. 429-439, 1 Oct.-Dec. 2016.

Paper III extends upon **Paper II** by showing both (a) how an integer linear programming (ILP) formulation can be derived from the placement constraints proposed in **Paper II** (**GP1**), and (b) under what circumstances the additional placement becomes too much of a computational burden for a placement optimization algorithm (**GP2**). Similarly, it addresses the overall research goal via both research objectives (**RO1** and **RO2**).

The results provide two main insights. The first is that the number of placement constraints affects the scheduler's ability to find a satisfying placement more severely than general background load. This is unsurprising because background load "only" makes the size of the knapsacks smaller (recall Section 4) while still letting the solver use them all as it sees fit.

The second is that for a high number of hosts with low capacity, component *affinity* is the dominant factor affecting ability to find a solution. This too is unsurprising because one would intuitively expect affinity to mean that larger items (e.g. two VMs with affinity) must be accommodated in a single knapsack, whereas anti-affinity just means that two specific components cannot be co-placed. Anti-affinity thus reduces the set of possible solutions slightly, which does not make the actual scheduling decision much more difficult to process.

Author contributions *As with the sibling paper (**Paper II**), Daniel (then with the last name Espling) and Lars co-formulated the core idea and co-implemented the translation of placement constraints into solver inputs. Lars then authored the majority of the paper. Crucial support and guidance on ILP was provided by Wubin Li, the group's expert on formulating scheduling in the form of an ILP problem. It should also be noted that the paper required submission to several conference venues and journals; this work (with slight modifications on each occasion) was done by Daniel Espling. As stated previously, it was agreed that overall* *credit for* ***Paper II*** *and* ***Paper III*** *should be shared equally between Daniel and Lars. The supervisors Johan Tordsson (Wubin's supervisor) and Erik Elmroth provided key feedback in the process of shepherding the paper to publication.*

6.5 Paper IV

L. Larsson, W. Tärneberg, C. Klein, and E. Elmroth, "Quality-Elasticity: Improved Resource Utilization, Throughput, and Response Times via Adjusting Output Quality to Current Operating Conditions", *2019 IEEE International Conference on Autonomic Computing (ICAC)*, Umeå, Sweden, 2019, pp. 52-62.

Paper IV addresses the research goal from the point of view of the Service Provider: if momentary resource scarcity is inevitable, how should such a situation be dealt with? Currently, software does not adjust itself in this situation — if a system is overloaded, programs simply run slower because there is more resource contention. This leads to a slower response, or, in the worst case, failure to respond within a predefined time frame, causing a timeout.

In **Paper IV**, we extend the ideas of Brownout [Kle+14a]. Brownout proposed adding a circuit breaker-like functionality to services that introduces a binary choice about whether to serve "optional content". In this context, optional content is content that is valuable for some reason (in the case of Brownout, product recommendations shown on product detail pages at an e-commerce site, which are known to increase sales [FH07; FH09]) but not strictly necessary.

Rather than the binary choice offered by Brownout, we propose a more granular approach: choosing not just *whether* to serve optional content, but also *how* it should be generated. Is there perhaps already cached content that can be used? Alternatively, can we offer less computationally intensive product recommendations by making cheaper database queries? And if resources are plentiful, can we perhaps choose computationally *more* expensive alternatives to generate higher-quality responses? Such decisions should be based on current conditions and resource availability, providing timely mitigation on a per-request timescale (**GP2**), and handled completely by the Service Provider (**RO1**).

Using the same simulator as Brownout, we introduce these new options and show that throughput is significantly increased, response times are accordingly lowered, and that more requests are served with at least *some* optional content rather than *none*. Obviously, the output quality of some of the served optional content is reduced to achieve these results, but if serving optional content increases profitability [FH07; FH09], overall profits should be higher even in these cases than they would be otherwise.

Author contributions *Lars performed the software and experimental design, implemented the simulator, conducted the experiments, and authored the paper. Lars had the core idea, and it was discussed extensively with William Tärneberg. William's continuous feedback and encouragement, along with that of the supervisors Cristian Klein and Erik Elmroth, greatly improved the paper.*

6.6 Paper V

L. Larsson, W. Tärneberg, C. Klein, E. Elmroth, and M. Kihl, "Impact of etcd Deployment on Kubernetes, Istio, and Application Performance", in *Software: Practice and Experience*, 2020.

Paper V is an experience paper that presents lessons learned while conducting a large number of experiments in a cloud environment. The key takeaway message is that the cloud software constituting the platform upon which applications are deployed is also complex and can suffer from resource scarcity. If that happens, even application performance can suffer. Thus, researchers must view the entire stack as the system under test, not just their application.

The etcd database is a key component of Kubernetes — every control plane decision is recorded in it, and it is under heavy I/O pressure due to a consistent and high-volume stream of requests. To ensure data durability, etcd makes heavy use of its backing hard drive storage. If the hard drive is "too slow" (whatever that means for a particular size of cluster and data churn level), etcd will operate too slowly, and in the worst case, there will be timeouts. These timeouts may be interpreted as failures by the Kubernetes control plane components, which can cause compounded failures. To understand why, we briefly explain some aspects of Kubernetes' functionality.

If a Kubernetes Pod fails its liveness tests, it should be replaced with a new Pod instance. When that happens, the new Pod instance should be listed as an Endpoint to the Service it belongs to, once readiness tests have been passed. Network requests to a Service can only be delegated to its listed Endpoints.

In this work, we created an application that could easily be overwhelmed by too many concurrent requests. Thus, when the request volume is sufficiently high, an application instance (Pod) will exhaust its memory allocation, causing it to be killed by the Linux kernel's out-of-memory (OOM) process killer. With the process killed, its corresponding liveness test will naturally fail. The Pod should then be replaced, as outlined above.

Thus, in the event of a high request volume (and thus a high load on a Service), rapidly replacing the failed Pod is vital to ensure that sufficient Endpoints are available to handle the overall load.

We found that if the backing storage of the etcd component is also under high load, the Pod replacement process fails to occur in a timely manner, which greatly affects the application's performance and (auto-scaling) behavior. Application performance and behavior in experiments where the etcd database was deployed with a (very slow) networked file system as its hard drive differed significantly from that observed when it stored its data on a (very fast) RAM-disk.

The takeaway message from this experience paper is that we must be aware of how our increasingly complex cloud platform software behaves and is deployed. Measuring only raw application performance can therefore be misleading if the platform is not also taken into account. This lesson informed the approach used for system evaluation in **Paper VII**, where instead of measuring only, e.g., network response times, we instead focused on counting messages and bytes sent across the network. This minimized non-deterministic disturbances due to control plane deployment.

Author contributions *Lars performed the vast majority of the work on this paper, including conceiving the initial idea, design, implementation, conducting experiments, and authorship. The work was performed in close collaboration with William Tärneberg, with whom the methodology and results were discussed thoroughly and over many iterations. William authored the paper's formal description of how queuing theory could be used to conduct performance testing. Valuable feedback was provided by the supervisors.*

6.7 Paper VI

L. Larsson, H. Gustafsson, C. Klein, and E. Elmroth, "Decentralized Kubernetes Federation Control Plane", *submitted* 2020.

In **Paper VI**, we return to the topic of federated cloud infrastructures (the subject of **Paper I**). In particular, we target geographically dispersed and numerous infrastructures at the network edge — the foundational infrastructures of edge computing. The problem domain differs from that of **Paper I** with respect to the sheer scale of the federations under consideration: the number of collaborating sites is orders of magnitude larger. Deploying applications to hundreds or thousands of Kubernetes clusters requires a more scalable control plane than that which currently exists.

In this position paper, we argue that a decentralized control plane is required to deal with the scale required for edge computing. To avoid problems relating to concurrent control plane updates caused by variation in resource availability over a massive federation, we propose the use of Conflict-free Replicated Data Types (CRDTs, recall Section 5.2) in a distributed database that spans the entire federation. All member clusters of the federation can independently volunteer to claim part of the overall required deployment, updating the shared tally in the distributed database. In this way, autonomy is preserved (**GP1**), and because the overall resource requirements are known on the global level of the system, scheduling can be done in parallel across all clusters, which should result in good distributed scheduling performance (**GP2**).

The proposed ideas would have to be implemented by Infrastructure Providers because we consider it unfeasible for Service Providers to maintain a list of all sites in a federation with hundreds or thousands of edge computing sites. Edge computing infrastructure must be managed by Infrastructure Providers; consequently, this paper relates to **RO2**.

The ideas presented in this position paper were and still are in very early developmental stages, and various possible implementations are being considered. This work was conducted while Lars was visiting Ericsson Research in Lund as a visiting researcher, and also resulted in **Patent II**.

Author contributions *Harald and Lars, as co-authors of the patent, believe that credit for the original idea should be split equally between them. Lars was given full responsibility for authorship of the academic position paper, and his supervisors Cristian and Erik provided valuable feedback.*

6.8 Paper VII

L. Larsson, W. Tärneberg, C. Klein, M. Kihl, and E. Elmroth, "Towards Soft Circuit Breaking in Service Meshes via Application-agnostic Caching", *submitted* 2020.

While the results of **Paper IV** were very promising, implementing them in practice would be rather labor-intensive and require the development and maintenance of several different implementations of specific functionalities with different resource requirements. Practically speaking, development costs would likely prohibit this for all but the most valuable business-critical systems.

Paper VII presents a more practical general solution in the form of a caching infrastructure that seamlessly and without requiring application modification can dynamically estimate how long cached previous responses will remain valid. Caching is rarely used in inter-service communication, which is surprising, given that the micro-service architecture design encourages separation of concerns and data ownership, potentially leading to frequent inter-service calls to obtain data that may not have changed since it was last used (see Section 5.3).

We evaluate the proposed system and the associated cache validity time estimation algorithms using two suites of experiments. First, we identify algorithm parametrizations that reduce the number of network requests without introducing too many errors due to data staleness. Second, we show that a real third-party application (Hipster Shop by Google Cloud Platform) can be deployed onto our caching infrastructure without modification, and that the identified conservative algorithm parametrizations manage to cache about 80% of requests, reducing total network traffic by 40%.

These results support our vision of seamlessly offering this type of functionality in service meshes as a "soft circuit breaker", i.e. one that does not just allow or disallow traffic between services but can reduce service load by using cached data where possible (**GP2**). This would greatly benefit edge computing, as research has shown that both poor data locality [Ngu+19; McC+19] and resource contention are hindrances to its adoption.

This work relates to research objectives **RO1** and **RO2** in that the proposed solution could be offered as a networking feature in a service mesh by the Infrastructure Provider, or be used directly by the Service Provider. Because caching in the manner described does not require application modification, both options are potentially viable.

Author contributions *The vast majority of the work was performed by Lars, including software and conceptual design, implementation, experiments, and authorship of the paper. Again, the work was conducted in collaboration with William Tärneberg, with whom the methodology and results were discussed thoroughly and over many iterations. Supervisors provided valuable feedback.*

Chapter 7
Conclusion

This thesis has a single research goal: to explore methods for managing cloud resource scarcity. This goal was addressed from two perspectives, resulting in two research objectives: to explore ways in which Service Providers and Infrastructure Providers can mitigate the adverse effects of resource scarcity on application or platform performance. The solutions presented in the papers following the kappa rest on three pillars: collaboration in inter-cloud federations, intelligent scheduling within and across federations, and making applications adapt to current resource availability.

On the basis of extensive literature reviews and personal experience gained over the many years spanned by this thesis (2009–2020), we have made the case that the cloud and the software deployed onto it have co-evolved over time and bred a new line of software that can truly be called cloud-native. This software has evolved to exploit the unique resource offerings of the cloud, turning what would previously have been seen as weaknesses of shared and dynamic infrastructure into strengths. Cloud-native software, by embracing the dynamic nature of cloud infrastructure, thus also helps Infrastructure Providers offer their services cost-efficiently.

The long period of time over which this thesis was written grants us the privilege of hindsight: we can see evidence of the older papers' impact on the research community and possibly also the cloud computing industry. **Paper I** has been cited over 85 times according to Google Scholar. The work done within the RESERVOIR project on the API that the author worked on directly and elaborated upon in **Paper I** influenced a multi-cloud compatibility API called Open Cloud Computing Interface (OCCI), which was subsequently standardized by the Open Grid Forum and has continued to evolve.

Regarding **Paper II** and **Paper III**, it is noteworthy that industry giants such as Amazon Web Services and Microsoft *now* offer a language to express cloud component affinity and anti-affinity rules today, but they did not before those papers were published. The more recent cloud-native container orchestrator Kubernetes also offers such a language and functionality. It is impossible to say whether any of these developments were (in-)directly influenced by our work — perhaps we just tapped into the *Zeitgeist* of the federated cloud computing field, but did so a few years before it had matured enough to be offered by public cloud vendors. Since RESERVOIR and its successor project OPTIMIS both deeply influenced EU research on cloud computing together with industry partners such as IBM and SAP, *some* influence should perhaps not be immediately excluded.

We expect that the complementary approaches presented in this thesis could be combined to further deepen the collaboration between Service Provider and Infrastructure Provider in delivering cloud-native applications to end users in ways that allow efficient sharing of resources and delivery of applications. Taken as a whole, the implementation of the ideas presented in this thesis would result in applications that can be efficiently scheduled and migrated across clouds (**Paper I** and **Paper VI**) without violating requirements relating to service structure or data locality (**Paper II** and **Paper III**), and that make better use of available resources wherever they are deployed (**Paper IV** and **Paper VII**). We have also shared our experience of conducting scientific research on cloud computing infrastructure (**Paper V**), which suggests that control plane software should perhaps also be able to adapt to current resource availability. Thus, our contributions have helped pave the way to more efficient use of available resources. Overall, the main conclusion drawn from the work presented herein is that cloud resource scarcity is best managed in a collaborative manner that respects both the autonomy and independence of all parties, as all parties benefit when optimal use can be made of the resources at our shared disposal.

Bibliography

[AB99] Tarek F Abdelzaher and Nina Bhatti. “Web content adaptation to improve server overload behavior”. In: *Computer Networks* 31.11 (May 17, 1999), pp. 1563–1577. ISSN: 1389-1286. DOI: `10.1016/S1389-1286(99)00031-6`. URL: `http://www.sciencedirect.com/science/article/pii/S1389128699000316` (visited on June 9, 2020).

[Aba12] Daniel Abadi. “Consistency Tradeoffs in Modern Distributed Database System Design: CAP is Only Part of the Story”. In: *Computer* 45.2 (Feb. 2012). Conference Name: Computer, pp. 37–42. ISSN: 1558-0814. DOI: `10.1109/MC.2012.33`.

[AJ99] Martin Arlitt and Tai Jin. *Workload Characterization of the 1998 World Cup Web Site*. HPL-1999-35(R.1). Internet Systems and Applications Laboratory, HP Laboratories Palo Alto, Sept. 1999.

[Ali+19] Ahmed Ali-Eldin, Jonathan Westin, Bin Wang, Prateek Sharma, and Prashant Shenoy. “SpotWeb: Running Latency-sensitive Distributed Web Services on Transient Cloud Servers”. In: *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*. HPDC ’19. Phoenix, AZ, USA: Association for Computing Machinery, June 2019, pp. 1–12. ISBN: 978-1-4503-6670-0. DOI: `10.1145/3307681.3325397`. URL: `https://doi.org/10.1145/3307681.3325397` (visited on June 3, 2020).

[Amb+19] Pradeep Ambati, David Irwin, Prashant Shenoy, Lixin Gao, Ahmed Ali-Eldin, and Jeannie Albrecht. “Understanding Synchronization Costs for Distributed ML on Transient Cloud Resources”. In: *2019 IEEE International Conference on Cloud Engineering (IC2E)*. June 2019, pp. 145–155. DOI: 10.1109/IC2E.2019.00029.

[Ans+09] Jason Ansel, Cy Chan, Yee Lok Wong, Marek Olszewski, Qin Zhao, Alan Edelman, and Saman Amarasinghe. “PetaBricks: a language and compiler for algorithmic choice”. In: *ACM SIGPLAN Notices* 44.6 (June 15, 2009), pp. 38–49. ISSN: 0362-1340. DOI: 10.1145/1543135.1542481. URL: https://doi.org/10.1145/1543135.1542481 (visited on June 9, 2020).

[Asg+12] Muhammad Rizwan Asghar, Mihaela Ion, Giovanni Russello, and Bruno Crispo. “Securing Data Provenance in the Cloud”. en. In: *Open Problems in Network Security*. Ed. by Jan Camenisch and Dogan Kesdogan. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2012, pp. 145–160. ISBN: 978-3-642-27585-2. DOI: 10.1007/978-3-642-27585-2_12.

[Att+06] Andrea Attanasio, Gianpaolo Ghiani, Lucio Grandinetti, and Francesca Guerriero. “Auction algorithms for decentralized parallel machine scheduling”. In: *Parallel Computing*. Optimization on Grids - Optimization for Grids 32.9 (Oct. 1, 2006), pp. 701–709. ISSN: 0167-8191. DOI: 10.1016/j.parco.2006.03.002. URL: http://www.sciencedirect.com/science/article/pii/S0167819106000597 (visited on June 22, 2020).

[AZH18] Mohammad Aazam, Sherali Zeadally, and Khaled A. Harras. “Fog Computing Architecture, Evaluation, and Future Research Directions”. In: *IEEE Communications Magazine* 56.5 (May 2018). Conference Name: IEEE Communications Magazine, pp. 46–52. ISSN: 1558-1896. DOI: 10.1109/MCOM.2018.1700707.

[Bar+20] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador Garcia, Sergio Gil-Lopez, Daniel Molina, Richard Benjamins, Raja Chatila, and Francisco Herrera. "Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI". en. In: *Information Fusion* 58 (June 2020), pp. 82–115. ISSN: 1566-2535. DOI: `10.1016/j.inffus.2019.12.012`. URL: `http://www.sciencedirect.com/science/article/pii/S1566253519308103` (visited on June 11, 2020).

[BCR09] Ken Birman, Gregory Chockler, and Robbert van Renesse. "Toward a cloud computing research agenda". In: *ACM SIGACT News* 40.2 (June 2009), pp. 68–80. ISSN: 0163-5700. DOI: `10.1145/1556154.1556172`. URL: `https://doi.org/10.1145/1556154.1556172` (visited on May 20, 2020).

[BD13] Philip A. Bernstein and Sudipto Das. "Rethinking eventual consistency". In: *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. SIGMOD '13. New York, New York, USA: Association for Computing Machinery, June 22, 2013, pp. 923–928. ISBN: 978-1-4503-2037-5. DOI: `10.1145/2463676.2465339`. URL: `https://doi.org/10.1145/2463676.2465339` (visited on June 8, 2020).

[BE11] David Breitgand and Amir Epstein. "SLA-aware placement of multi-virtual machine elastic services in compute clouds". In: *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops*. ISSN: 1573-0077. May 2011, pp. 161–168. DOI: `10.1109/INM.2011.5990687`.

[BG13] Peter BAILIS and Ali GHODSI. "Eventual Consistency Today: Limitations, Extensions, and Beyond". In: *Eventual Consistency Today: Limitations, Extensions, and Beyond* 56.5 (2013). Num Pages: 9 Place: New York, NY

Publisher: Association for Computing Machinery, pp. 55–63. ISSN: 0001-0782.

[BH07] Luiz André Barroso and Urs Hölzle. “The Case for Energy-Proportional Computing”. In: *IEEE Computer* 40 (2007). URL: http://www.computer.org/portal/site/computer/index.jsp?pageID=computer_level1&path=computer/homepage/Dec07&file=feature.xml&xsl=article.xsl (visited on June 10, 2020).

[Bia16] Randy Bias. *The History of Pets vs Cattle and How to Use the Analogy Properly*. Sept. 2016. URL: http://cloudscaling.com/blog/cloud-computing/the-history-of-pets-vs-cattle/ (visited on June 11, 2020).

[Bie+12] Annette Bieniusa, Marek Zawirski, Nuno Preguiça, Marc Shapiro, Carlos Baquero, Valter Balegas, and Sérgio Duarte. “An optimized conflict-free replicated set”. In: *arXiv:1210.3368 [cs]* (Oct. 11, 2012). arXiv: 1210.3368. URL: http://arxiv.org/abs/1210.3368 (visited on June 8, 2020).

[Bir+14] I Bird, P Buncic, F Carminati, M Cattaneo, P Clarke, I Fisk, M Girone, J Harvey, B Kersevan, P Mato, R Mount, and B Panzer-Steindel. *Update of the Computing Models of the WLCG and the LHC Experiments*. Tech. rep. CERN-LHCC-2014-014. LCG-TDR-002. Apr. 2014. URL: https://cds.cern.ch/record/1695401.

[Bir07] Ken Birman. “The promise, and limitations, of gossip protocols”. In: *ACM SIGOPS Operating Systems Review* 41.5 (Oct. 1, 2007), pp. 8–13. ISSN: 0163-5980. DOI: 10.1145/1317379.1317382. URL: https://doi.org/10.1145/1317379.1317382 (visited on June 8, 2020).

[BKR10] David Breitgand, Gilad Kutiel, and Danny Raz. “Cost-aware live migration of services in the cloud”. en. In: *Proceedings of the 3rd Annual Haifa Experimental Systems Conference on - SYSTOR ’10*. Haifa, Israel: ACM Press, 2010,

p. 1. ISBN: 978-1-60558-908-4. DOI: 10.1145/1815695.1815709. URL: http://portal.acm.org/citation.cfm?doid=1815695.1815709 (visited on June 4, 2020).

[BPV09] Rajkumar Buyya, Suraj Pandey, and Christian Vecchiola. “Cloudbus Toolkit for Market-Oriented Cloud Computing”. en. In: *Cloud Computing*. Ed. by Martin Gilje Jaatun, Gansen Zhao, and Chunming Rong. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2009, pp. 24–44. ISBN: 978-3-642-10665-1. DOI: 10.1007/978-3-642-10665-1_4.

[Bre00] Eric A. Brewer. “Towards robust distributed systems”. ACM Symposium on Principles of Distributed Computing (PODC). Portland, Oregon, USA, July 16, 2000. URL: https://doi.org/10.1145/343477.343502 (visited on June 8, 2020).

[Bre12a] Eric Brewer. “CAP twelve years later: How the "rules" have changed”. In: *Computer* 45.2 (Feb. 2012). Conference Name: Computer, pp. 23–29. ISSN: 1558-0814. DOI: 10.1109/MC.2012.37.

[Bre12b] Eric Brewer. “Pushing the CAP: Strategies for Consistency and Availability”. In: *Computer* 45.2 (Feb. 1, 2012), pp. 23–29. ISSN: 0018-9162. DOI: 10.1109/MC.2012.37. URL: https://doi.org/10.1109/MC.2012.37 (visited on June 8, 2020).

[BT11] David Bermbach and Stefan Tai. “Eventual consistency: How soon is eventual? An evaluation of Amazon S3’s consistency behavior”. In: *Proceedings of the 6th Workshop on Middleware for Service Oriented Computing*. MW4SOC ’11. Lisbon, Portugal: Association for Computing Machinery, Dec. 12, 2011, pp. 1–6. ISBN: 978-1-4503-1067-3. DOI: 10.1145/2093185.2093186. URL: https://doi.org/10.1145/2093185.2093186 (visited on June 8, 2020).

[Bur06] Mike Burrows. “The Chubby lock service for loosely-coupled distributed systems”. In: *7th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 2006.

[Bur14] Sebastian Burckhardt. *Principles of Eventual Consistency*. Principles of Eventual Consistency. Vol. 1. Foundations and Trends® in Programming Languages. Now Publishers, Oct. 2014. 1-150. URL: `https://www.microsoft.com/en-us/research/publication/principles-of-eventual-consistency/`.

[Bye17] Charles C. Byers. “Architectural Imperatives for Fog Computing: Use Cases, Requirements, and Architectural Techniques for Fog-Enabled IoT Networks”. In: *IEEE Communications Magazine* 55.8 (Aug. 2017). Conference Name: IEEE Communications Magazine, pp. 14–20. ISSN: 1558-1896. DOI: `10.1109/MCOM.2017.1600885`.

[BYV08] Rajkumar Buyya, Chee Shin Yeo, and Srikumar Venugopal. “Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities”. In: *2008 10th IEEE International Conference on High Performance Computing and Communications*. Sept. 2008, pp. 5–13. DOI: `10.1109/HPCC.2008.172`.

[Cha+06] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. “Bigtable: A Distributed Storage System for Structured Data”. In: *7th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 2006, pp. 205–218.

[Che+09] Betty H. C. Cheng, Rogério de Lemos, Holger Giese, Paola Inverardi, Jeff Magee, Jesper Andersson, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, Giovanna Di Marzo Serugendo, Schahram Dustdar, Anthony Finkelstein, Cristina Gacek, Kurt Geihs, Vincenzo Grassi, Gabor Karsai, Holger M. Kienle, Jeff Kramer, Marin Litoiu, Sam Malek, Raffaela Mirandola, Hausi A. Müller, Sooyong Park, Mary Shaw, Matthias Tichy, Massimo Tivoli, Danny Weyns, and Jon Whittle. "Software Engineering for Self-Adaptive Systems: A Research Roadmap". In: *Software Engineering for Self-Adaptive Systems*. Ed. by Betty H. C. Cheng, Rogério de Lemos, Holger Giese, Paola Inverardi, and Jeff Magee. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2009, pp. 1–26. ISBN: 978-3-642-02161-9. DOI: 10.1007/978-3-642-02161-9_1. URL: https://doi.org/10.1007/978-3-642-02161-9_1 (visited on June 9, 2020).

[Chi+12] Houssem-Eddine Chihoub, Shadi Ibrahim, Gabriel Antoniu, and María S. Pérez. "Harmony: Towards Automated Self-Adaptive Consistency in Cloud Storage". In: *2012 IEEE International Conference on Cluster Computing*. 2012 IEEE International Conference on Cluster Computing. ISSN: 2168-9253. Sept. 2012, pp. 293–301. DOI: 10.1109/CLUSTER.2012.56.

[Chi+13] Vinay K. Chippa, Srimat T. Chakradhar, Kaushik Roy, and Anand Raghunathan. "Analysis and characterization of inherent application resilience for approximate computing". In: *Proceedings of the 50th Annual Design Automation Conference*. DAC '13. Austin, Texas: Association for Computing Machinery, May 29, 2013, pp. 1–9. ISBN: 978-1-4503-2071-9. DOI: 10.1145/2463209.2488873. URL: https://doi.org/10.1145/2463209.2488873 (visited on June 8, 2020).

[Cis18] Cisco Systems Inc. "Cisco Annual Internet Report (2018–2023) White Paper". Visited 14 February 2020. 2018. URL: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html.

[CJC19] *Ericsson Mobility Report November 2019*. en. Tech. rep. EMR-November-2019. Ericsson AB, Nov. 2019.

[CLC13] Jui-Hao Chiang, Han-Lin Li, and Tzi-cker Chiueh. "Working Set-based Physical Memory Ballooning". In:

10th International Conference on Autonomic Computing ({ICAC} 13). 2013, pp. 95–99. ISBN: 978-1-931971-02-7. URL: `https://www.usenix.org/conference/icac13/technical-sessions/presentation/chiang` (visited on June 22, 2020).

[CMD62] Fernando J. Corbató, Marjorie Merwin-Daggett, and Robert C. Daley. "An experimental time-sharing system". In: *Proceedings of the May 1-3, 1962, spring joint computer conference*. AIEE-IRE '62 (Spring). San Francisco, California: Association for Computing Machinery, May 1962, pp. 335–344. ISBN: 978-1-4503-7875-8. DOI: `10.1145/1460833.1460871`. URL: `https://doi.org/10.1145/1460833.1460871` (visited on May 17, 2020).

[Com18] CNCF Technical Oversight Committee. *CNCF Cloud Native Definition v1.0*. en. Library Catalog: github.com. June 2018. URL: `https://github.com/cncf/toc/blob/master/DEFINITION.md` (visited on May 20, 2020).

[CP17] Emiliano Casalicchio and Vanessa Perciballi. "Measuring Docker Performance: What a Mess!!!" In: *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*. ICPE '17 Companion. L&apos;Aquila, Italy: Association for Computing Machinery, Apr. 18, 2017, pp. 11–16. ISBN: 978-1-4503-4899-7. DOI: `10.1145/3053600.3053605`. URL: `https://doi.org/10.1145/3053600.3053605` (visited on May 26, 2020).

[CSK02] Duk-Ho Chang, Jin Hyun Son, and Myoung Ho Kim. "Critical path identification in the context of a workflow". In: *Information and Software Technology* 44.7 (May 15, 2002), pp. 405–417. ISSN: 0950-5849. DOI: `10.1016/S0950-5849(02)00025-3`. URL: `http://www.sciencedirect.com/science/article/pii/S0950584902000253` (visited on June 8, 2020).

[CZS17] Songqing Chen, Tao Zhang, and Weisong Shi. "Fog Computing". In: *IEEE Internet Computing* 21.2 (Mar. 2017).

Conference Name: IEEE Internet Computing, pp. 4–6. ISSN: 1941-0131. DOI: 10.1109/MIC.2017.39.

[Dem+87] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. "Epidemic algorithms for replicated database maintenance". In: *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*. PODC '87. Vancouver, British Columbia, Canada: Association for Computing Machinery, Dec. 1, 1987, pp. 1–12. ISBN: 978-0-89791-239-6. DOI: 10.1145/41840.41841. URL: https://doi.org/10.1145/41840.41841 (visited on June 8, 2020).

[DG08] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters". en. In: *Communications of the ACM* 51.1 (Jan. 2008), pp. 107–113. ISSN: 0001-0782, 1557-7317. DOI: 10.1145/1327452.1327492. URL: https://dl.acm.org/doi/10.1145/1327452.1327492 (visited on June 10, 2020).

[DR97] Pedro C. Diniz and Martin C. Rinard. "Dynamic feedback: an effective technique for adaptive computing". In: *Proceedings of the ACM SIGPLAN 1997 conference on Programming language design and implementation*. PLDI '97. Las Vegas, Nevada, USA: Association for Computing Machinery, May 1, 1997, pp. 71–84. ISBN: 978-0-89791-907-4. DOI: 10.1145/258915.258923. URL: https://doi.org/10.1145/258915.258923 (visited on June 9, 2020).

[Dre08] Ulrich Drepper. "The Cost of Virtualization". In: *Queue* 6.1 (Jan. 2008), pp. 28–35. ISSN: 1542-7730. DOI: 10.1145/1348583.1348591. URL: https://doi.org/10.1145/1348583.1348591 (visited on May 18, 2020).

[ES18] Melike Erol-Kantarci and Sukhmani Sukhmani. "Caching and Computing at the Edge for Mobile Augmented Reality and Virtual Reality (AR/VR) in 5G". en. In: *Ad Hoc Networks*. Ed. by Yifeng Zhou and Thomas Kunz. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Cham: Springer International Publishing, 2018, pp. 169–177. ISBN: 978-3-319-74439-1. DOI: 10.1007/978-3-319-74439-1_15.

[Fer+12] Ana Juan Ferrer, Francisco Hernández, Johan Tordsson, Erik Elmroth, Ahmed Ali-Eldin, Csilla Zsigri, Raül Sirvent, Jordi Guitart, Rosa M. Badia, Karim Djemame, Wolfgang Ziegler, Theo Dimitrakos, Srijith K. Nair, George Kousiouris, Kleopatra Konstanteli, Theodora Varvarigou, Benoit Hudzia, Alexander Kipp, Stefan Wesner, Marcelo Corrales, Nikolaus Forgó, Tabassum Sharif, and Craig Sheridan. "OPTIMIS: A holistic approach to cloud service provisioning". en. In: *Future Generation Computer Systems* 28.1 (Jan. 2012), pp. 66–77. ISSN: 0167-739X. DOI: 10.1016/j.future.2011.05.022. URL: http://www.sciencedirect.com/science/article/pii/S0167739X1100104X (visited on June 2, 2020).

[FH07] Daniel M. Fleder and Kartik Hosanagar. "Recommender systems and their impact on sales diversity". In: *Proceedings of the 8th ACM conference on Electronic commerce*. EC '07. San Diego, California, USA: Association for Computing Machinery, June 11, 2007, pp. 192–199. ISBN: 978-1-59593-653-0. DOI: 10.1145/1250910.1250939. URL: https://doi.org/10.1145/1250910.1250939 (visited on May 8, 2020).

[FH09] Daniel Fleder and Kartik Hosanagar. "Blockbuster Culture's Next Rise or Fall: The Impact of Recommender Systems on Sales Diversity". In: *Management Science* 55.5 (Mar. 6, 2009). Publisher: INFORMS, pp. 697–712. ISSN: 0025-1909. DOI: 10.1287/mnsc.1080.0974. URL: https://pubsonline.informs.org/doi/abs/10.1287/mnsc.1080.0974 (visited on May 8, 2020).

[Flo+18] Luciano Floridi, Josh Cowls, Monica Beltrametti, Raja Chatila, Patrice Chazerand, Virginia Dignum, Christoph Luetge, Robert Madelin, Ugo Pagallo, Francesca Rossi, Burkhard Schafer, Peggy Valcke, and Effy Vayena.

"AI4People—An Ethical Framework for a Good AI Society: Opportunities, Risks, Principles, and Recommendations". en. In: *Minds and Machines* 28.4 (Dec. 2018), pp. 689–707. ISSN: 1572-8641. DOI: 10.1007/s11023-018-9482-5. URL: https://doi.org/10.1007/s11023-018-9482-5 (visited on June 11, 2020).

[Fos+08] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. "Cloud Computing and Grid Computing 360-Degree Compared". In: *2008 Grid Computing Environments Workshop*. ISSN: 2152-1093. Nov. 2008, pp. 1–10. DOI: 10.1109/GCE.2008.4738445.

[Fos02] Ian Foster. "What is the grid? a three point checklist". In: *GRID today* 1.6 (2002), pp. 32–36.

[Fos03] Ian Foster. "THE GRID: Computing without Bounds". In: *Scientific American* 288.4 (2003). Publisher: Scientific American, a division of Nature America, Inc., pp. 78–85. ISSN: 0036-8733. URL: https://www.jstor.org/stable/26060247 (visited on May 15, 2020).

[Fra+12] Eitan Frachtenberg, Dan Lee, Marco Magarelli, Veerendra Mulay, and Jay Park. "Thermal design in the open compute datacenter". In: *13th InterSociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems*. ISSN: 1087-9870. May 2012, pp. 530–538. DOI: 10.1109/ITHERM.2012.6231476.

[Fra12] E. Frachtenberg. "Holistic Datacenter Design in the Open Compute Project". In: *Computer* 45.07 (July 2012), pp. 83–85. ISSN: 1558-0814. DOI: 10.1109/MC.2012.235.

[GA02] Sanny Gustavsson and Sten F. Andler. "Self-stabilization and eventual consistency in replicated real-time databases". In: *Proceedings of the first workshop on Self-healing systems*. WOSS '02. Charleston, South Carolina: Association for Computing Machinery, Nov. 18, 2002, pp. 105–107. ISBN: 978-1-58113-609-8. DOI: 10.1145/582128.582150. URL: https:

//doi.org/10.1145/582128.582150 (visited on June 8, 2020).

[Gar99] Simson Garfinkel. *Architects of the Information Society: 35 Years of the Laboratory for Computer Science at MIT*. en. Google-Books-ID: Fc7dkLGLKrcC. MIT Press, 1999. ISBN: 978-0-262-07196-3.

[GGT10] Inigo Goiri, Jordi Guitart, and Jordi Torres. “Characterizing Cloud Federation for Enhancing Providers’ Profit”. In: *2010 IEEE 3rd International Conference on Cloud Computing*. ISSN: 2159-6190. July 2010, pp. 123–130. DOI: 10.1109/CLOUD.2010.32.

[Gog+12] Spyridon V. Gogouvitis, George Kousiouris, George Vafiadis, Elliot K. Kolodner, and Dimosthenis Kyriazis. “OPTIMIS and VISION Cloud: How to Manage Data in Clouds”. en. In: *Euro-Par 2011: Parallel Processing Workshops*. Ed. by Michael Alexander, Pasqua D’Ambra, Adam Belloum, George Bosilca, Mario Cannataro, Marco Danelutto, Beniamino Di Martino, Michael Gerndt, Emmanuel Jeannot, Raymond Namyst, Jean Roman, Stephen L. Scott, Jesper Larsson Traff, Geoffroy Vallée, and Josef Weidendorfer. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2012, pp. 35–44. ISBN: 978-3-642-29737-3. DOI: 10.1007/978-3-642-29737-3_5.

[Gog+13] S. V. Gogouvitis, M. C. Jaeger, H. Kolodner, D. Kyriazis, F. Longo, M. Lorenz, A. Messina, M. Montagnuolo, E. Salant, and F. Tusa. “Vision Cloud: A Cloud Storage Solution Supporting Modern Media Production”. In: *SMPTE Motion Imaging Journal* 122.7 (Oct. 2013). Conference Name: SMPTE Motion Imaging Journal, pp. 30–37. ISSN: 2160-2492. DOI: 10.5594/j18341.

[Goi+15] Inigo Goiri, Ricardo Bianchini, Santosh Nagarakatte, and Thu D. Nguyen. “ApproxHadoop: Bringing Approximations to MapReduce Frameworks”. In: *ACM SIGPLAN Notices* 50.4 (Mar. 14, 2015), pp. 383–397. ISSN: 0362-1340. DOI: 10.1145/2775054.2694351. URL: https:

//doi.org/10.1145/2775054.2694351 (visited on Apr. 27, 2020).

[Goo15] Tom Goodwin. *The Battle Is For The Customer Interface*. Mar. 2015. URL: https://techcrunch.com/2015/03/03/in-the-age-of-disintermediation-the-battle-is-all-for-the-customer-interface/ (visited on June 15, 2020).

[Gor+12] Abel Gordon, Nadav Amit, Nadav Har'El, Muli Ben-Yehuda, Alex Landau, Assaf Schuster, and Dan Tsafrir. "ELI: bare-metal performance for I/O virtualization". In: *ACM SIGPLAN Notices* 47.4 (Mar. 2012), pp. 411–422. ISSN: 0362-1340. DOI: 10.1145/2248487.2151020. URL: https://doi.org/10.1145/2248487.2151020 (visited on May 18, 2020).

[HK19] Michael Haenlein and Andreas Kaplan. "A Brief History of Artificial Intelligence: On the Past, Present, and Future of Artificial Intelligence". en. In: *California Management Review* 61.4 (Aug. 2019). Publisher: SAGE Publications Inc, pp. 5–14. ISSN: 0008-1256. DOI: 10.1177/0008125619864925. URL: https://doi.org/10.1177/0008125619864925 (visited on June 11, 2020).

[HO13] Jie Han and Michael Orshansky. "Approximate computing: An emerging paradigm for energy-efficient design". In: *2013 18th IEEE European Test Symposium (ETS)*. 2013 18th IEEE European Test Symposium (ETS). ISSN: 1558-1780. May 2013, pp. 1–6. DOI: 10.1109/ETS.2013.6569370.

[HR83] Theo Haerder and Andreas Reuter. "Principles of transaction-oriented database recovery". In: *ACM Computing Surveys* 15.4 (Dec. 2, 1983), pp. 287–317. ISSN: 0360-0300. DOI: 10.1145/289.291. URL: https://doi.org/10.1145/289.291 (visited on June 8, 2020).

[Hua+13] Ye Huang, Nik Bessis, Peter Norrington, Pierre Kuonen, and Beat Hirsbrunner. "Exploring decentralized dynamic scheduling for grids and clouds using the community-aware scheduling algorithm". In: *Future Generation Computer Systems*. Including Special section: AIRCC-NetCoM 2009 and Special section: Clouds and Service-Oriented Architectures 29.1 (Jan. 1, 2013), pp. 402–415. ISSN: 0167-739X. DOI: `10.1016/j.future.2011.05.006`. URL: `http://www.sciencedirect.com/science/article/pii/S0167739X11000872` (visited on June 22, 2020).

[Ios+11] Alexandru Iosup, Simon Ostermann, M. Nezih Yigitbasi, Radu Prodan, Thomas Fahringer, and Dick Epema. "Performance Analysis of Cloud Computing Services for Many-Tasks Scientific Computing". In: *IEEE Transactions on Parallel and Distributed Systems* 22.6 (June 2011). Conference Name: IEEE Transactions on Parallel and Distributed Systems, pp. 931–945. ISSN: 1558-2183. DOI: `10.1109/TPDS.2011.66`.

[Irw+19] David Irwin, Prashant Shenoy, Pradeep Ambati, Prateek Sharma, Supreeth Shastri, and Ahmed Ali-Eldin. "The Price Is (Not) Right: Reflections on Pricing for Transient Cloud Servers". In: *2019 28th International Conference on Computer Communication and Networks (ICCCN)*. ISSN: 2637-9430. July 2019, pp. 1–9. DOI: `10.1109/ICCCN.2019.8846933`.

[Ism+15] Marcelo Alexandre da Cruz Ismael, Cesar Alberto da Silva, Gabriel Costa Silva, and Reginaldo Ré. "An Empirical Study for Evaluating the Performance of jclouds". In: *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*. Nov. 2015, pp. 115–122. DOI: `10.1109/CloudCom.2015.61`.

[IYE11] Alexandru Iosup, Nezih Yigitbasi, and Dick Epema. "On the Performance Variability of Production Cloud Services". In: *2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. May 2011, pp. 104–113. DOI: `10.1109/CCGrid.2011.22`.

[JHB01] K. Jenkins, K. Hopkinson, and K. Birman. "A gossip protocol for subgroup multicast". In: *Proceedings 21st International Conference on Distributed Computing Systems*

Workshops. Proceedings 21st International Conference on Distributed Computing Systems Workshops. Apr. 2001, pp. 25–30. DOI: 10.1109/CDCS.2001.918682.

[Jin+09] Hai Jin, Li Deng, Song Wu, Xuanhua Shi, and Xiaodong Pan. "Live virtual machine migration with adaptive, memory compression". In: *2009 IEEE International Conference on Cluster Computing and Workshops*. ISSN: 2168-9253. Aug. 2009, pp. 1–10. DOI: 10.1109/CLUSTR.2009.5289170.

[Joy15] Ann Mary Joy. "Performance comparison between Linux containers and virtual machines". In: *2015 International Conference on Advances in Computer Engineering and Applications*. Mar. 2015, pp. 342–346. DOI: 10.1109/ICACEA.2015.7164727.

[JP13] Raj Jain and Subharthi Paul. "Network virtualization and software defined networking for cloud computing: a survey". In: *IEEE Communications Magazine* 51.11 (Nov. 2013). Conference Name: IEEE Communications Magazine, pp. 24–31. ISSN: 1558-1896. DOI: 10.1109/MCOM.2013.6658648.

[Kep05] Jeffrey O. Kephart. "Research challenges of autonomic computing". In: *Proceedings of the 27th international conference on Software engineering*. ICSE '05. St. Louis, MO, USA: Association for Computing Machinery, May 15, 2005, pp. 15–22. ISBN: 978-1-58113-963-1. DOI: 10.1145/1062455.1062464. URL: https://doi.org/10.1145/1062455.1062464 (visited on June 9, 2020).

[Kha+19] Wazir Zada Khan, Ejaz Ahmed, Saqib Hakak, Ibrar Yaqoob, and Arif Ahmed. "Edge computing: A survey". en. In: *Future Generation Computer Systems* 97 (Aug. 2019), pp. 219–235. ISSN: 0167-739X. DOI: 10.1016/j.future.2019.02.050. URL: http://www.sciencedirect.com/science/article/pii/S0167739X18319903 (visited on June 3, 2020).

[Kim+19] Dongmin Kim, Hanif Muhammad, Eunsam Kim, Sumi Helal, and Choonhwa Lee. “TOSCA-Based and Federation-Aware Cloud Orchestration for Kubernetes Container Platform”. en. In: *Applied Sciences* 9.1 (Jan. 2019). Number: 1 Publisher: Multidisciplinary Digital Publishing Institute, p. 191. DOI: `10.3390/app9010191`. URL: `https://www.mdpi.com/2076-3417/9/1/191` (visited on June 2, 2020).

[Kle+14a] Cristian Klein, Martina Maggio, Karl-Erik Årzén, and Francisco Hernández-Rodriguez. “Brownout: building more robust cloud applications”. In: *Proceedings of the 36th International Conference on Software Engineering*. ICSE 2014. Hyderabad, India: Association for Computing Machinery, May 31, 2014, pp. 700–711. ISBN: 978-1-4503-2756-5. DOI: `10.1145/2568225.2568227`. URL: `https://doi.org/10.1145/2568225.2568227` (visited on June 8, 2020).

[Kle+14b] Cristian Klein, Alessandro Vittorio Papadopoulos, Manfred Dellkrantz, Jonas Dürango, Martina Maggio, Karl-Erik Årzén, Francisco Hernández-Rodriguez, and Erik Elmroth. “Improving Cloud Service Resilience Using Brownout-Aware Load-Balancing”. In: *2014 IEEE 33rd International Symposium on Reliable Distributed Systems*. 2014 IEEE 33rd International Symposium on Reliable Distributed Systems. ISSN: 1060-9857. Oct. 2014, pp. 31–40. DOI: `10.1109/SRDS.2014.14`.

[KM07] Jeff Kramer and Jeff Magee. “Self-Managed Systems: an Architectural Challenge”. In: *Future of Software Engineering (FOSE ’07)*. Future of Software Engineering (FOSE ’07). May 2007, pp. 259–268. DOI: `10.1109/FOSE.2007.19`.

[KMZ15] Kenji E. Kushida, Jonathan Murray, and John Zysman. “Cloud Computing: From Scarcity to Abundance”. en. In: *Journal of Industry, Competition and Trade* 15.1 (Mar. 2015), pp. 5–19. ISSN: 1573-7012. DOI: `10.1007/`

s10842-014-0188-y. URL: https://doi.org/10.1007/s10842-014-0188-y (visited on May 20, 2020).

[KQ17] Nane Kratzke and Peter-Christian Quint. "Understanding cloud-native applications after 10 years of cloud computing - A systematic mapping study". en. In: *Journal of Systems and Software* 126 (Apr. 2017), pp. 1–16. ISSN: 0164-1212. DOI: 10.1016/j.jss.2017.01.001. URL: http://www.sciencedirect.com/science/article/pii/S0164121217300018 (visited on May 19, 2020).

[Krz+18] Jakub Krzywda, Ahmed Ali-Eldin, Eddie Wadbro, Per-Olov Östberg, and Erik Elmroth. "ALPACA: Application Performance Aware Server Power Capping". In: *2018 IEEE International Conference on Autonomic Computing (ICAC)*. ISSN: 2474-0756. Sept. 2018, pp. 41–50. DOI: 10.1109/ICAC.2018.00014.

[KS15] Sowmya Karunakaran and Rangaraja P. Sundarraj. "Bidding Strategies for Spot Instances in Cloud Computing Markets". In: *IEEE Internet Computing* 19.3 (May 2015). Conference Name: IEEE Internet Computing, pp. 32–40. ISSN: 1941-0131. DOI: 10.1109/MIC.2014.87.

[KW14] Ryan K.L. Ko and Mark A. Will. "Progger: An Efficient, Tamper-Evident Kernel-Space Logger for Cloud Data Provenance Tracking". In: *2014 IEEE 7th International Conference on Cloud Computing*. ISSN: 2159-6190. June 2014, pp. 881–889. DOI: 10.1109/CLOUD.2014.121.

[LC16] Philipp Leitner and Jürgen Cito. "Patterns in the Chaos—A Study of Performance Variation and Predictability in Public IaaS Clouds". In: *ACM Transactions on Internet Technology* 16.3 (Apr. 2016), 15:1–15:23. ISSN: 1533-5399. DOI: 10.1145/2885497. URL: https://doi.org/10.1145/2885497 (visited on May 25, 2020).

[Lee16] Craig A. Lee. "Cloud Federation Management and Beyond: Requirements, Relevant Standards, and Gaps". In: *IEEE Cloud Computing* 3.1 (Jan. 2016). Conference Name:

IEEE Cloud Computing, pp. 42–49. ISSN: 2325-6095. DOI: 10.1109/MCC.2016.15.

[Lia+17] Xueping Liang, Sachin Shetty, Deepak Tosh, Charles Kamhoua, Kevin Kwiat, and Laurent Njilla. "ProvChain: A Blockchain-Based Data Provenance Architecture in Cloud Environment with Enhanced Privacy and Availability". In: *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID).* May 2017, pp. 468–477. DOI: 10.1109/CCGRID.2017.8.

[Llo+14] Wyatt Lloyd, Michael J. Freedman, Michael Kaminsky, and David G. Andersen. "Don't Settle for Eventual Consistency". In: *Queue* 12.3 (Mar. 15, 2014), pp. 30–45. ISSN: 1542-7730. DOI: 10.1145/2602649.2610533. URL: https://doi.org/10.1145/2602649.2610533 (visited on June 8, 2020).

[Lv+17] Xiao Lv, Fazhi He, Weiwei Cai, and Yuan Cheng. "A string-wise CRDT algorithm for smart and large-scale collaborative editing systems". In: *Advanced Engineering Informatics* 33 (Aug. 1, 2017), pp. 397–409. ISSN: 1474-0346. DOI: 10.1016/j.aei.2016.10.005. URL: http://www.sciencedirect.com/science/article/pii/S1474034616301811 (visited on June 8, 2020).

[Man+18] Johannes Manner, Martin Endreß, Tobias Heckel, and Guido Wirtz. "Cold Start Influencing Factors in Function as a Service". In: *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion).* Dec. 2018, pp. 181–188. DOI: 10.1109/UCC-Companion.2018.00054.

[MB17] Pavel Mach and Zdenek Becvar. "Mobile Edge Computing: A Survey on Architecture and Computation Offloading". In: *IEEE Communications Surveys Tutorials* 19.3 (2017). Conference Name: IEEE Communications Surveys Tutorials, pp. 1628–1656. ISSN: 1553-877X. DOI: 10.1109/COMST.2017.2682318.

[McC+19] Jonathan McChesney, Nan Wang, Ashish Tanwer, Eyal de Lara, and Blesson Varghese. "DeFog: fog computing benchmarks". In: *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*. SEC '19. Arlington, Virginia: Association for Computing Machinery, Nov. 2019, pp. 47–58. ISBN: 978-1-4503-6733-2. DOI: `10.1145/3318216.3363299`. URL: `https://doi.org/10.1145/3318216.3363299` (visited on May 4, 2020).

[McK+08] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. "OpenFlow: enabling innovation in campus networks". In: *ACM SIGCOMM Computer Communication Review* 38.2 (Mar. 2008), pp. 69–74. ISSN: 0146-4833. DOI: `10.1145/1355734.1355746`. URL: `https://doi.org/10.1145/1355734.1355746` (visited on May 18, 2020).

[MD11] Michele Mazzucco and Marlon Dumas. "Achieving Performance and Availability Guarantees with Spot Instances". In: *2011 IEEE International Conference on High Performance Computing and Communications*. Sept. 2011, pp. 296–303. DOI: `10.1109/HPCC.2011.46`.

[Mer+11] Arif Merchant, Mustafa Uysal, Pradeep Padala, Xiaoyun Zhu, Sharad Singhal, and Kang Shin. "Maestro: quality-of-service in large disk arrays". In: *Proceedings of the 8th ACM international conference on Autonomic computing*. ICAC '11. Karlsruhe, Germany: Association for Computing Machinery, June 14, 2011, pp. 245–254. ISBN: 978-1-4503-0607-2. DOI: `10.1145/1998582.1998638`. URL: `https://doi.org/10.1145/1998582.1998638` (visited on June 9, 2020).

[Mer14] Dirk Merkel. "Docker: lightweight Linux containers for consistent development and deployment". In: *Linux Journal* 2014.239 (Mar. 2014), 2:2. ISSN: 1075-3583.

[MF01] A.W. Mu'alem and D.G. Feitelson. "Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling". In: *IEEE Transactions on Parallel and Distributed Systems* 12.6 (June 2001). Conference Name: IEEE Transactions on Parallel and Distributed Systems, pp. 529–543. ISSN: 1558-2183. DOI: 10.1109/71.932708.

[MG11] Peter Mell and Tim Grance. *The NIST Definition of Cloud Computing*. en. Tech. rep. NIST Special Publication (SP) 800-145. National Institute of Standards and Technology, Sept. 2011. DOI: https://doi.org/10.6028/NIST.SP.800-145. URL: https://csrc.nist.gov/publications/detail/sp/800-145/final (visited on May 15, 2020).

[Mit16] Sparsh Mittal. "A Survey of Techniques for Approximate Computing". In: *ACM Computing Surveys* 48.4 (Mar. 18, 2016), 62:1–62:33. ISSN: 0360-0300. DOI: 10.1145/2893356. URL: https://doi.org/10.1145/2893356 (visited on June 8, 2020).

[MLM11] Dejan Milojičić, Ignacio M. Llorente, and Ruben S. Montero. "OpenNebula: A Cloud Management Tool". In: *IEEE Internet Computing* 15.2 (Mar. 2011). Conference Name: IEEE Internet Computing, pp. 11–14. ISSN: 1941-0131. DOI: 10.1109/MIC.2011.44.

[Moh+19] Anup Mohan, Harshad Sane, Kshitij Doshi, Saikrishna Edupuganti, Naren Nayak, and Vadim Sukhomlinov. "Agile Cold Starts for Scalable Serverless". In: *11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19)*. Renton, WA: USENIX Association, July 2019. URL: https://www.usenix.org/conference/hotcloud19/presentation/mohan.

[Mou+18] Carla Mouradian, Diala Naboulsi, Sami Yangui, Roch H. Glitho, Monique J. Morrow, and Paul A. Polakos. "A Comprehensive Survey on Fog Computing: State-of-the-Art and Research Challenges". In: *IEEE Communications Surveys Tutorials* 20.1 (2018). Conference Name: IEEE Communications Surveys Tutorials, pp. 416–464. ISSN: 1553-877X. DOI: 10.1109/COMST.2017.2771153.

[MS70] R. A. Meyer and L. H. Seawright. “A virtual machine time-sharing system”. In: *IBM Systems Journal* 9.3 (1970). Conference Name: IBM Systems Journal, pp. 199–218. ISSN: 0018-8670. DOI: 10.1147/sj.93.0199.

[Nah03] Fiona Nah. “A Study on Tolerable Waiting Time: How Long Are Web Users Willing to Wait?” In: vol. 23. Jan. 2003, p. 285. DOI: 10.1080/01449290410001669914.

[Nel09] Michael Nelson. “Virtual machine migration”. US7484208B1. Library Catalog: Google Patents. Jan. 2009. URL: https://patents.google.com/patent/US7484208B1/en (visited on May 18, 2020).

[Ngu+19] Chanh Nguyen, Amardeep Mehta, Cristian Klein, and Erik Elmroth. “Why cloud applications are not ready for the edge (yet)”. In: *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*. SEC ’19. Arlington, Virginia: Association for Computing Machinery, Nov. 2019, pp. 250–263. ISBN: 978-1-4503-6733-2. DOI: 10.1145/3318216.3363298. URL: https://doi.org/10.1145/3318216.3363298 (visited on May 8, 2020).

[Nur+09] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov. “The Eucalyptus Open-Source Cloud-Computing System”. In: *2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*. May 2009, pp. 124–131. DOI: 10.1109/CCGRID.2009.93.

[Osa+17] Opeyemi Osanaiye, Shuo Chen, Zheng Yan, Rongxing Lu, Kim-Kwang Raymond Choo, and Mqhele Dlodlo. “From Cloud to Fog Computing: A Review and a Conceptual Live VM Migration Framework”. In: *IEEE Access* 5 (2017). Conference Name: IEEE Access, pp. 8284–8300. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2017.2692960.

[Ost+10] Simon Ostermann, Alexandria Iosup, Nezih Yigitbasi, Radu Prodan, Thomas Fahringer, and Dick Epema. "A Performance Analysis of EC2 Cloud Computing Services for Scientific Computing". en. In: *Cloud Computing*. Ed. by Dimiter R. Avresky, Michel Diaz, Arndt Bode, Bruno Ciciani, and Eliezer Dekel. Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering. Berlin, Heidelberg: Springer, 2010, pp. 115–131. ISBN: 978-3-642-12636-9. DOI: `10.1007/978-3-642-12636-9_9`.

[OST14] Justice Opara-Martins, Reza Sahandi, and Feng Tian. "Critical review of vendor lock-in and its impact on adoption of cloud computing". In: *International Conference on Information Society (i-Society 2014)*. Nov. 2014, pp. 92–97. DOI: `10.1109/i-Society.2014.7009018`.

[Pag+19] Ugo Pagallo, Paola Aurucci, Pompeu Casanovas, Raja Chatila, Patrice Chazerand, Virginia Dignum, Christoph Luetge, Robert Madelin, Burkhard Schafer, and Peggy Valcke. *AI4People - On Good AI Governance: 14 Priority Actions, a S.M.A.R.T. Model of Governance, and a Regulatory Toolbox*. en. SSRN Scholarly Paper ID 3486508. Rochester, NY: Social Science Research Network, Nov. 2019. URL: `https://papers.ssrn.com/abstract=3486508` (visited on June 11, 2020).

[Pap+17] Alessandro Vittorio Papadopoulos, Jakub Krzywda, Erik Elmroth, and Martina Maggio. "Power-aware cloud brownout: Response time and power consumption control". In: *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. 2017 IEEE 56th Annual Conference on Decision and Control (CDC). Dec. 2017, pp. 2686–2691. DOI: `10.1109/CDC.2017.8264049`.

[Pap+19] Alessandro Vittorio Papadopoulos, Laurens Versluis, André Bauer, Nikolas Herbst, Jóakim Von Kistowski, Ahmed Ali-eldin, Cristina Abad, José Nelson Amaral, Petr Tůma, and Alexandru Iosup. "Methodological Principles for Reproducible Performance Evaluation in Cloud

Computing". In: *IEEE Transactions on Software Engineering* (2019). Conference Name: IEEE Transactions on Software Engineering, pp. 1–1. ISSN: 1939-3520. DOI: 10.1109/TSE.2019.2927908.

[PDT18] Gopika Premsankar, Mario Di Francesco, and Tarik Taleb. "Edge Computing for the Internet of Things: A Case Study". In: *IEEE Internet of Things Journal* 5.2 (Apr. 2018). Conference Name: IEEE Internet of Things Journal, pp. 1275–1284. ISSN: 2327-4662. DOI: 10.1109/JIOT.2018.2805263.

[Phi+10] Jeremy Philippe, Noel De Palma, Fabienne Boyer, and et Olivier Gruber. "Self-adaptation of service level in distributed systems". In: *Software: Practice and Experience* 40.3 (2010). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.957, pp. 259–283. ISSN: 1097-024X. DOI: 10.1002/spe.957. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.957 (visited on June 9, 2020).

[PMA17] Carlo Puliafito, Enzo Mingozzi, and Giuseppe Anastasi. "Fog Computing for the Internet of Mobile Things: Issues and Challenges". In: *2017 IEEE International Conference on Smart Computing (SMARTCOMP)*. May 2017, pp. 1–6. DOI: 10.1109/SMARTCOMP.2017.7947010.

[Pog+14] Nicolas Poggi, David Carrera, Ricard Gavaldà, Eduard Ayguadé, and Jordi Torres. "A methodology for the evaluation of high response time on E-commerce users and sales". In: *Information Systems Frontiers* 16.5 (Nov. 1, 2014), pp. 867–885. ISSN: 1572-9419. DOI: 10.1007/s10796-012-9387-4. URL: https://doi.org/10.1007/s10796-012-9387-4 (visited on June 8, 2020).

[Pro+18] Andrew Prout, William Arcand, David Bestor, Bill Bergeron, Chansup Byun, Vijay Gadepally, Michael Houle, Matthew Hubbell, Michael Jones, Anna Klein, Peter Michaleas, Lauren Milechin, Julie Mullen, Antonio Rosa, Siddharth Samsi, Charles Yee, Albert Reuther, and Jeremy Kepner. "Measuring the Impact of Spectre and

Meltdown". In: *2018 IEEE High Performance extreme Computing Conference (HPEC)*. ISSN: 2377-6943. Sept. 2018, pp. 1–5. DOI: `10.1109/HPEC.2018.8547554`.

[PSK15] Bryan Pon, Timo Seppälä, and Martin Kenney. "One Ring to Unite Them All: Convergence, the Smartphone, and the Cloud". en. In: *Journal of Industry, Competition and Trade* 15.1 (Mar. 2015), pp. 21–33. ISSN: 1573-7012. DOI: `10.1007/s10842-014-0189-x`. URL: `https://doi.org/10.1007/s10842-014-0189-x` (visited on May 20, 2020).

[Quo+18] Do Le Quoc, Istemi Ekin Akkus, Pramod Bhatotia, Spyros Blanas, Ruichuan Chen, Christof Fetzer, and Thorsten Strufe. "ApproxJoin: Approximate Distributed Joins". In: *Proceedings of the ACM Symposium on Cloud Computing*. SoCC '18. Carlsbad, CA, USA: Association for Computing Machinery, Oct. 11, 2018, pp. 426–438. ISBN: 978-1-4503-6011-1. DOI: `10.1145/3267809.3267834`. URL: `https://doi.org/10.1145/3267809.3267834` (visited on Apr. 27, 2020).

[Red18] Red Hat, Inc. *Understanding cloud-native apps*. en. Library Catalog: www.redhat.com. June 2018. URL: `https://www.redhat.com/en/topics/cloud-native-apps` (visited on May 20, 2020).

[Roc+09a] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, M. Ben-Yehuda, W. Emmerich, and F. Galan. "The Reservoir model and architecture for open federated cloud computing". In: *IBM Journal of Research and Development* 53.4 (July 2009). Conference Name: IBM Journal of Research and Development, 4:1–4:11. ISSN: 0018-8646. DOI: `10.1147/JRD.2009.5429058`.

[Roc+09b] B. Rochwerger, A. Galis, E. Levy, J. A. Caceres, D. Breitgand, Y. Wolfsthal, I. M. Llorente, M. Wusthoff, R. S. Montero, and E. Elmroth. "RESERVOIR: Management technologies and requirements for next generation Service Oriented Infrastructures". In: *2009 IFIP/IEEE In-*

ternational Symposium on Integrated Network Management. ISSN: 1573-0077. June 2009, pp. 307–310. DOI: 10.1109/INM.2009.5188828.

[Roc+11] Benny Rochwerger, David Breitgand, Amir Epstein, David Hadas, Irit Loy, Kenneth Nagin, Johan Tordsson, Carmelo Ragusa, Massimo Villari, Stuart Clayman, Eliezer Levy, Alessandro Maraschini, Philippe Massonet, Henar Muñoz, and Giovanni Tofetti. "Reservoir - When One Cloud Is Not Enough". In: *Computer* 44.3 (Mar. 2011). Conference Name: Computer, pp. 44–51. ISSN: 1558-0814. DOI: 10.1109/MC.2011.64.

[Ros14] Rami Rosen. "Linux containers and the future cloud". In: *Linux Journal* 2014.240 (Apr. 2014), 3:3. ISSN: 1075-3583.

[Roy+15] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. "Inside the Social Network's (Datacenter) Network". In: *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. SIGCOMM '15. London, United Kingdom: Association for Computing Machinery, Aug. 2015, pp. 123–137. ISBN: 978-1-4503-3542-3. DOI: 10.1145/2785956.2787472. URL: https://doi.org/10.1145/2785956.2787472 (visited on June 10, 2020).

[RWH11] Charles Reiss, John Wilkes, and Joseph L. Hellerstein. *Google cluster-usage traces: format + schema*. Technical Report. Revised 2014-11-17 for version 2.1. Posted at https://github.com/google/cluster-data. Mountain View, CA, USA: Google Inc., Nov. 2011.

[SAE12] Omar Sefraoui, Mohammed Aissaoui, and Mohsine Eleuldj. "OpenStack: Toward an Open-source Solution for Cloud Computing". en. In: *International Journal of Computer Applications* 55.3 (Oct. 2012), pp. 38–42. ISSN: 09758887. DOI: 10.5120/8738-2991. URL: http://research.ijcaonline.org/volume55/number3/pxc3882991.pdf (visited on June 2, 2020).

[SAS19] Prateek Sharma, Ahmed Ali-Eldin, and Prashant Shenoy. "Resource Deflation: A New Approach For Transient Resource Reclamation". In: *Proceedings of the Fourteenth EuroSys Conference 2019*. EuroSys '19. Dresden, Germany: Association for Computing Machinery, Mar. 2019, pp. 1–17. ISBN: 978-1-4503-6281-8. DOI: `10.1145/3302424.3303945`. URL: `https://doi.org/10.1145/3302424.3303945` (visited on June 3, 2020).

[Sat+13] Benjamin Satzger, Waldemar Hummer, Christian Inzinger, Philipp Leitner, and Schahram Dustdar. "Winds of Change: From Vendor Lock-In to the Meta Cloud". In: *IEEE Internet Computing* 17.1 (Jan. 2013). Conference Name: IEEE Internet Computing, pp. 69–73. ISSN: 1941-0131. DOI: `10.1109/MIC.2013.19`.

[SB10] Mohsen Amini Salehi and Rajkumar Buyya. "Adapting Market-Oriented Scheduling Policies for Cloud Computing". en. In: *Algorithms and Architectures for Parallel Processing*. Ed. by Ching-Hsien Hsu, Laurence T. Yang, Jong Hyuk Park, and Sang-Soo Yeo. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2010, pp. 351–362. ISBN: 978-3-642-13119-6. DOI: `10.1007/978-3-642-13119-6_31`.

[SH11] David Schneider and Quentin Hardy. "Under the hood at Google and Facebook". In: *IEEE Spectrum* 48.6 (June 2011). Conference Name: IEEE Spectrum, pp. 63–67. ISSN: 1939-9340. DOI: `10.1109/MSPEC.2011.5779795`.

[Sha+11a] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. *A comprehensive study of Convergent and Commutative Replicated Data Types*. report. Inria – Centre Paris-Rocquencourt ; INRIA, Jan. 13, 2011, p. 50. URL: `https://hal.inria.fr/inria-00555588` (visited on June 8, 2020).

[Sha+11b] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. "Conflict-Free Replicated Data Types". In: *Stabilization, Safety, and Security of Distributed Systems*. Ed. by Xavier Défago, Franck Petit, and Vincent Villain.

Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2011, pp. 386–400. ISBN: 978-3-642-24550-3. DOI: 10.1007/978-3-642-24550-3_29.

[Shi+16] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. “Edge Computing: Vision and Challenges”. In: *IEEE Internet of Things Journal* 3.5 (Oct. 2016). Conference Name: IEEE Internet of Things Journal, pp. 637–646. ISSN: 2327-4662. DOI: 10.1109/JIOT.2016.2579198.

[Sim56] H. A. Simon. “Rational choice and the structure of the environment”. In: *Psychological Review* 63.2 (Mar. 1956), pp. 129–138. ISSN: 0033-295X. DOI: 10.1037/h0042769.

[SK17] Marc Shapiro and Bettina Kemme. *Eventual Consistency*. Pages: 2. Springer, June 2017. ISBN: 978-1-4899-7993-3. DOI: 10.1007/978-1-4899-7993-3_1366-2. URL: https://hal.inria.fr/hal-01547451 (visited on June 8, 2020).

[SRC13] Gabriel Costa Silva, Louis M. Rose, and Radu Calinescu. “Towards a Model-Driven Solution to the Vendor Lock-In Problem in Cloud Computing”. In: *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*. Vol. 1. Dec. 2013, pp. 711–716. DOI: 10.1109/CloudCom.2013.131.

[Sri+02] S. Srinivasan, R. Kettimuthu, V. Subramani, and P. Sadayappan. “Characterization of backfilling strategies for parallel job scheduling”. In: *Proceedings. International Conference on Parallel Processing Workshop*. ISSN: 1530-2016. Aug. 2002, pp. 514–519. DOI: 10.1109/ICPPW.2002.1039773.

[SRS17] Michael Schneider, Jason Rambach, and Didier Stricker. “Augmented reality based on edge computing using the example of remote live support”. In: *2017 IEEE International Conference on Industrial Technology (ICIT)*. Mar. 2017, pp. 1277–1282. DOI: 10.1109/ICIT.2017.7915547.

[SS13] Navin Sabharwal and Ravi Shankar. *Apache CloudStack Cloud Computing*. Packt Publishing, 2013. ISBN: 978-1-78216-010-6.

[Ste20] Bijan Stephen. *A pizzeria owner made money buying his own $24 pizzas from DoorDash for $16*. en. Library Catalog: www.theverge.com. May 2020. URL: `https://www.theverge.com/2020/5/18/21262316/doordash-pizza-profits-venture-capital-the-margins-ranjan-roy` (visited on June 10, 2020).

[Sue+13] Chun Hui Suen, Ryan K.L. Ko, Yu Shyang Tan, Peter Jagadpramana, and Bu Sung Lee. "S2Logger: End-to-End Data Tracking Mechanism for Cloud Data Provenance". In: *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*. ISSN: 2324-9013. July 2013, pp. 594–602. DOI: `10.1109/TrustCom.2013.73`.

[Svä+11] Petter Svärd, Benoit Hudzia, Johan Tordsson, and Erik Elmroth. "Evaluation of delta compression techniques for efficient live migration of large virtual machines". In: *Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*. VEE '11. Newport Beach, California, USA: Association for Computing Machinery, Mar. 2011, pp. 111–120. ISBN: 978-1-4503-0687-4. DOI: `10.1145/1952682.1952698`. URL: `https://doi.org/10.1145/1952682.1952698` (visited on May 18, 2020).

[TEF07] Dan Tsafrir, Yoav Etsion, and Dror G. Feitelson. "Backfilling Using System-Generated Predictions Rather than User Runtime Estimates". In: *IEEE Transactions on Parallel and Distributed Systems* 18.6 (June 2007). Conference Name: IEEE Transactions on Parallel and Distributed Systems, pp. 789–803. ISSN: 1558-2183. DOI: `10.1109/TPDS.2007.70606`.

[TLE16] Luis Tomás, Ewnetu Bayuh Lakew, and Erik Elmroth. "Service Level and Performance Aware Dynamic Resource Allocation in Overbooked Data Centers". In:

2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid). May 2016, pp. 42–51. DOI: `10.1109/CCGrid.2016.29`.

[Tof+17] Giovanni Toffetti, Sandro Brunner, Martin Blöchlinger, Josef Spillner, and Thomas Michael Bohnert. "Self-managing cloud-native applications: Design, implementation, and experience". en. In: *Future Generation Computer Systems* 72 (July 2017), pp. 165–179. ISSN: 0167-739X. DOI: `10.1016/j.future.2016.09.002`. URL: `http://www.sciencedirect.com/science/article/pii/S0167739X16302977` (visited on May 19, 2020).

[Tom+14] Luis Tomás, Cristian Klein, Johan Tordsson, and Francisco Hernández-Rodríguez. "The Straw that Broke the Camel's Back: Safe Cloud Overbooking with Application Brownout". In: *2014 International Conference on Cloud and Autonomic Computing*. 2014 International Conference on Cloud and Autonomic Computing. Sept. 2014, pp. 151–160. DOI: `10.1109/ICCAC.2014.10`.

[Tor+12] Johan Tordsson, Rubén S. Montero, Rafael Moreno-Vozmediano, and Ignacio M. Llorente. "Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers". en. In: *Future Generation Computer Systems* 28.2 (Feb. 2012), pp. 358–367. ISSN: 0167-739X. DOI: `10.1016/j.future.2011.07.003`. URL: `http://www.sciencedirect.com/science/article/pii/S0167739X11001373` (visited on June 2, 2020).

[Tsi+01] Mark Tsimelzon, Bill Weihl, Joseph Chung, Dan Frantz, John Basso, Chris Newton, Mark Hale, Larry Jacobs, and Conleth O'Connell. *ESI Language Specification 1.0*. Ed. by Mark Nottingham. Aug. 2001. URL: `https://www.w3.org/TR/2001/NOTE-esi-lang-20010804` (visited on May 25, 2020).

[TT13] Luis Tomás and Johan Tordsson. "Improving cloud infrastructure utilization through overbooking". In: *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference*. CAC '13. Miami, Florida, USA: Association for Computing Machinery, Aug. 2013, pp. 1–10. ISBN: 978-1-4503-2172-3. DOI: 10.1145/2494621.2494627. URL: https://doi.org/10.1145/2494621.2494627 (visited on June 3, 2020).

[TT14] Luis Tomás and Johan Tordsson. "An Autonomic Approach to Risk-Aware Data Center Overbooking". In: *IEEE Transactions on Cloud Computing* 2.3 (July 2014). Conference Name: IEEE Transactions on Cloud Computing, pp. 292–305. ISSN: 2168-7161. DOI: 10.1109/TCC.2014.2326166.

[TW84] Van Tilborg and Wittie. "Wave Scheduling—Decentralized Scheduling of Task Forces in Multicomputers". In: *IEEE Transactions on Computers* C-33.9 (Sept. 1984). Conference Name: IEEE Transactions on Computers, pp. 835–844. ISSN: 1557-9956. DOI: 10.1109/TC.1984.1676500.

[TYL12] ShaoJie Tang, Jing Yuan, and Xiang-Yang Li. "Towards Optimal Bidding Strategy for Amazon EC2 Cloud Spot Instance". In: *2012 IEEE Fifth International Conference on Cloud Computing*. ISSN: 2159-6190. June 2012, pp. 91–98. DOI: 10.1109/CLOUD.2012.134.

[Ull75] J. D. Ullman. "NP-complete scheduling problems". en. In: *Journal of Computer and System Sciences* 10.3 (June 1975), pp. 384–393. ISSN: 0022-0000. DOI: 10.1016/S0022-0000(75)80008-0. URL: http://www.sciencedirect.com/science/article/pii/S0022000075800080 (visited on June 15, 2020).

[Var10] Jinesh Varia. *Migrating your Existing Applications to the AWS Cloud*. en. Oct. 2010. URL: https://media.amazonwebservices.com/CloudMigration-main.pdf.

[Váz+13] Carlos Vázquez, Luis Tomás, Ginés Moreno, and Johan Tordsson. "A Fuzzy Approach to Cloud Admission Control for Safe Overbooking". en. In: *Fuzzy Logic and Applications*. Ed. by Francesco Masulli, Gabriella Pasi, and Ronald Yager. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2013, pp. 212–225. ISBN: 978-3-319-03200-9. DOI: 10.1007/978-3-319-03200-9_22.

[VB18] Blesson Varghese and Rajkumar Buyya. “Next generation cloud computing: New trends and research directions”. en. In: *Future Generation Computer Systems* 79 (Feb. 2018), pp. 849–861. ISSN: 0167-739X. DOI: 10.1016/j.future.2017.09.020. URL: http://www.sciencedirect.com/science/article/pii/S0167739X17302224 (visited on May 20, 2020).

[Ver+15] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. “Large-scale cluster management at Google with Borg”. In: *Proceedings of the Tenth European Conference on Computer Systems*. EuroSys ’15. Bordeaux, France: Association for Computing Machinery, Apr. 2015, pp. 1–17. ISBN: 978-1-4503-3238-5. DOI: 10.1145/2741948.2741964. URL: https://doi.org/10.1145/2741948.2741964 (visited on June 3, 2020).

[Vil+12] David Villegas, Norman Bobroff, Ivan Rodero, Javier Delgado, Yanbin Liu, Aditya Devarakonda, Liana Fong, S. Masoud Sadjadi, and Manish Parashar. “Cloud federation in a layered service model”. en. In: *Journal of Computer and System Sciences*. JCSS Special Issue: Cloud Computing 2011 78.5 (Sept. 2012), pp. 1330–1344. ISSN: 0022-0000. DOI: 10.1016/j.jcss.2011.12.017. URL: http://www.sciencedirect.com/science/article/pii/S0022000011001620 (visited on June 2, 2020).

[Vog08] Werner Vogels. “Beyond Server Consolidation”. In: *Queue* 6.1 (Jan. 2008), pp. 20–26. ISSN: 1542-7730. DOI: 10.1145/1348583.1348590. URL: https://doi.org/10.1145/1348583.1348590 (visited on June 3, 2020).

[Voo+09] William Voorsluys, James Broberg, Srikumar Venugopal, and Rajkumar Buyya. “Cost of Virtual Machine Live Migration in Clouds: A Performance Evaluation”. en. In:

Cloud Computing. Ed. by Martin Gilje Jaatun, Gansen Zhao, and Chunming Rong. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2009, pp. 254–265. ISBN: 978-3-642-10665-1. DOI: `10.1007/978-3-642-10665-1_23`.

[Vou09] Mladen A. Vouk. "Cloud Computing – Issues, Research and Implementations". en-US. In: *CIT. Journal of Computing and Information Technology* 16.4 (June 2009). Number: 4, pp. 235–246. ISSN: 1846-3908. DOI: `10.2498/cit.1001391`. URL: `http://cit.fer.hr/index.php/CIT/article/view/1674` (visited on May 20, 2020).

[VT16] Rath Vannithamby and Shilpa Talwar. *Towards 5G: Applications, Requirements and Candidate Technologies*. en. Google-Books-ID: g2lxDQAAQBAJ. John Wiley & Sons, Nov. 2016. ISBN: 978-1-118-97991-4.

[VVB10] Ruben Van den Bossche, Kurt Vanmechelen, and Jan Broeckhove. "Cost-Optimal Scheduling in Hybrid IaaS Clouds for Deadline Constrained Workloads". In: *2010 IEEE 3rd International Conference on Cloud Computing*. 2010 IEEE 3rd International Conference on Cloud Computing. ISSN: 2159-6190. July 2010, pp. 228–235. DOI: `10.1109/CLOUD.2010.58`.

[Wan+18] Liang Wang, Mengyuan Li, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. "Peeking Behind the Curtains of Serverless Platforms". In: *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. Boston, MA: USENIX Association, July 2018, pp. 133–146. ISBN: ISBN 978-1-939133-01-4. URL: `https://www.usenix.org/conference/atc18/presentation/wang-liang`.

[Wel+01] Michael P. Wellman, William E. Walsh, Peter R. Wurman, and Jeffrey K. MacKie-Mason. "Auction Protocols for Decentralized Scheduling". In: *Games and Economic Behavior* 35.1 (Apr. 1, 2001), pp. 271–303. ISSN: 0899-8256. DOI: `10.1006/game.2000.0822`. URL: `http://www.sciencedirect.com/science/article/pii/S0899825600908224` (visited on June 22, 2020).

[Wil11] John Wilkes. *More Google cluster data*. Google research blog. Posted at http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html. Mountain View, CA, USA, Nov. 2011.

[Wil20a] John Wilkes. *Google cluster-usage traces v3*. Technical Report. Posted at https://github.com/google/cluster-data/blob/master/ClusterData2019.md. Mountain View, CA, USA: Google Inc., Apr. 2020.

[Wil20b] John Wilkes. *Yet more Google compute cluster trace data*. Google research blog. Posted at https://ai.googleblog.com/2020/04/yet-more-google-compute-cluster-trace.html. Mountain View, CA, USA, Apr. 2020.

[XB19] Minxian Xu and Rajkumar Buyya. "Brownout Approach for Adaptive Management of Resources and Applications in Cloud Computing Systems: A Taxonomy and Future Directions". In: *ACM Computing Surveys* 52.1 (Jan. 25, 2019), 8:1–8:27. ISSN: 0360-0300. DOI: 10.1145/3234151. URL: https://doi.org/10.1145/3234151 (visited on June 8, 2020).

[XDB16] Minxian Xu, Amir Vahid Dastjerdi, and Rajkumar Buyya. "Energy Efficient Scheduling of Cloud Application Components with Brownout". In: *IEEE Transactions on Sustainable Computing* 1.2 (July 2016). Conference Name: IEEE Transactions on Sustainable Computing, pp. 40–53. ISSN: 2377-3782. DOI: 10.1109/TSUSC.2017.2661339.

[XMK16] Qiang Xu, Todd Mytkowicz, and Nam Sung Kim. "Approximate Computing: A Survey". In: *IEEE Design Test* 33.1 (Feb. 2016). Conference Name: IEEE Design Test, pp. 8–22. ISSN: 2168-2364. DOI: 10.1109/MDAT.2015.2505723.

[XTB19] Minxian Xu, Adel Nadjaran Toosi, and Rajkumar Buyya. "iBrownout: An Integrated Approach for Managing Energy and Brownout in Container-Based Clouds". In: *IEEE Transactions on Sustainable Computing* 4.1 (Jan. 2019).

Conference Name: IEEE Transactions on Sustainable Computing, pp. 53–66. ISSN: 2377-3782. DOI: 10.1109/TSUSC.2018.2808493.

[Xu+14] Fei Xu, Fangming Liu, Linghui Liu, Hai Jin, Bo Li, and Baochun Li. "iAware: Making Live Migration of Virtual Machines Interference-Aware in the Cloud". In: *IEEE Transactions on Computers* 63.12 (Dec. 2014). Conference Name: IEEE Transactions on Computers, pp. 3012–3025. ISSN: 1557-9956. DOI: 10.1109/TC.2013.185.

[YAK12] Sangho Yi, Artur Andrzejak, and Derrick Kondo. "Monetary Cost-Aware Checkpointing and Migration on Amazon Cloud Spot Instances". In: *IEEE Transactions on Services Computing* 5.4 (2012). Conference Name: IEEE Transactions on Services Computing, pp. 512–524. ISSN: 1939-1374. DOI: 10.1109/TSC.2011.44.

[Yao+12] Yuan Yao, Longbo Huang, Abhihshek Sharma, Leana Golubchik, and Michael Neely. "Data centers power reduction: A two time scale approach for delay tolerant workloads". In: *2012 Proceedings IEEE INFOCOM*. 2012 Proceedings IEEE INFOCOM. ISSN: 0743-166X. Mar. 2012, pp. 1431–1439. DOI: 10.1109/INFCOM.2012.6195508.

[Yaz+17] Amir Yazdanbakhsh, Divya Mahajan, Hadi Esmaeilzadeh, and Pejman Lotfi-Kamran. "AxBench: A Multiplatform Benchmark Suite for Approximate Computing". In: *IEEE Design Test* 34.2 (Apr. 2017). Conference Name: IEEE Design Test, pp. 60–68. ISSN: 2168-2364. DOI: 10.1109/MDAT.2016.2630270.

[YKA10] Sangho Yi, Derrick Kondo, and Artur Andrzejak. "Reducing Costs of Spot Instances via Checkpointing in the Amazon Elastic Compute Cloud". In: *2010 IEEE 3rd International Conference on Cloud Computing*. ISSN: 2159-6190. July 2010, pp. 236–243. DOI: 10.1109/CLOUD.2010.35.

[YLL15] Shanhe Yi, Cheng Li, and Qun Li. "A Survey of Fog Computing: Concepts, Applications and Issues". In: *Proceedings of the 2015 Workshop on Mobile Big Data*. Mobidata '15. Hangzhou, China: Association for Computing Machinery, June 2015, pp. 37–42. ISBN: 978-1-4503-3524-9. DOI: 10.1145/2757384.2757397. URL: https://doi.org/10.1145/2757384.2757397 (visited on June 1, 2020).

[You+19] Ashkan Yousefpour, Caleb Fung, Tam Nguyen, Krishna Kadiyala, Fatemeh Jalali, Amirreza Niakanlahiji, Jian Kong, and Jason P. Jue. "All one needs to know about fog computing and related edge computing paradigms: A complete survey". en. In: *Journal of Systems Architecture* 98 (Sept. 2019), pp. 289–330. ISSN: 1383-7621. DOI: 10.1016/j.sysarc.2019.02.009. URL: http://www.sciencedirect.com/science/article/pii/S1383762118306349 (visited on June 2, 2020).

[Yu12] Weihai Yu. "A string-wise CRDT for group editing". In: *Proceedings of the 17th ACM international conference on Supporting group work*. GROUP '12. Sanibel Island, Florida, USA: Association for Computing Machinery, Oct. 27, 2012, pp. 141–144. ISBN: 978-1-4503-1486-2. DOI: 10.1145/2389176.2389198. URL: https://doi.org/10.1145/2389176.2389198 (visited on June 8, 2020).

[Zub19] Shoshana Zuboff. *The Age of Surveillance Capitalism*. PublicAffairs (Hachette Book Group), Jan. 2019. ISBN: 9781610395700.