

UMEÅ UNIVERSITY

Keeping Clouds Afloat

Oliver Larsson

Doctoral Thesis, November 2026
Department of Computing Science
Umeå University
Sweden

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

olars@cs.umu.se
`https://thesis.larssonoliver.com`

Copyright © 2026 Oliver Larsson
Except Paper I, © 2023 IEEE. Reprinted, with permission, from [1].
Paper II, © 2025, The Author(s). Reprinted, with permission, from [2].
Paper III, © 2025 IEEE. Reprinted, with permission, from [3].
Paper IV, © 2026, The Author(s).
Paper V, © 2026 IEEE. Reprinted, with permission, from [4].
Paper VI, © 2026, The Author(s).
Paper VII, © 2026, The Author(s).

ISBN **978-91-6850-127-7** (print)
978-91-6850-128-4 (digital)
ISSN **0348-0542**
UMINF **26.17**

Cover illustration by Oliver Larsson.

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 by Scandinavian Print Group, Hägersten, 2026.

"The best you can do is solve the problems of today based on what you learned yesterday. Keep learning and stay humble."

— Kelsey Hightower

Abstract

The proliferation of data-intensive applications and the emergence of artificial intelligence (AI) workloads have driven a rapid increase in demand for computing resources. At the same time, increasing heterogeneity in hardware configurations and workload behaviors has made resource management increasingly challenging, as complex interdependencies between hardware and software can lead to congestion in shared resources. Such congestion degrades workload performance and increases energy consumption, forcing resource owners to leave hardware underutilized to mitigate the issue. This presents a significant challenge to the sustainability and efficiency of modern computing infrastructures.

This thesis addresses the challenges of resource management in heterogeneous cloud computing environments, with a particular focus on understanding and mitigating resource congestion and its effects on workload performance and cluster efficiency. It considers resource management at three levels of the cloud computing stack: the individual workload, the compute node, and the entire cluster. By investigating the interactions between workloads and hardware resources while considering the needs of different stakeholders, the thesis develops insights and approaches for improving resource utilization and cluster performance.

Across seven academic papers, this thesis progressively develops an understanding of resource congestion and its implications for cloud resource management. It first investigates the complex interdependencies between shared memory subsystem resources and workload performance. It then explores mechanisms for controlling these resources to mitigate the negative effects of congestion. Building on this foundation, the thesis introduces a novel observability framework that quantifies the impact of resource congestion on workload performance at runtime, providing the information needed to reason about resource behavior at the workload level. Finally, these insights are brought together in a congestion-aware and intent-driven approach to resource management that considers both the state of the underlying infrastructure and the needs of different stakeholders. The thesis concludes by discussing how these principles can inform resource management in increasingly heterogeneous cloud computing environments and the challenges that remain for the next generation of distributed computing systems.

Sammanfattning

Den ökande förekomsten av dataintensiva applikationer och framväxten av artificiell intelligens (AI) har lett till en snabb ökning av efterfrågan på beräkningsresurser. Samtidigt har en växande heterogenitet i hårdvara och arbetslasters beteenden gjort resurshantering allt mer utmanande, eftersom komplexa beroenden mellan hårdvara och mjukvara kan leda till överbelastning av delade resurser. Sådan överbelastning försämrar arbetslasters prestanda och ökar energiförbrukningen, vilket tvingar resursägare att låta hårdvara gå underutnyttjad. Detta utgör en betydande utmaning för hållbarheten och effektiviteten hos moderna beräkningsinfrastrukturer.

Denna avhandling behandlar utmaningarna inom resurshantering i heterogena molnbaserade datormiljöer, med särskilt fokus på att förstå och motverka resursöverbelastning och dess effekter på arbetslasters prestanda och klustereffektivitet. Resurshantering studeras på tre nivåer i molnets stack: den individuella arbetslasten, beräkningsnoden och klustret. Genom att undersöka samspelet mellan arbetslaster och hårdvaruresurser, samtidigt som olika intressenters behov beaktas, presenterar avhandlingen insikter och metoder för att förbättra resursutnyttjande och prestanda.

Genom sju vetenskapliga artiklar utvecklar avhandlingen successivt en förståelse för resursöverbelastning och dess konsekvenser för resurshantering i molnmiljöer. Inledningsvis undersöks de komplexa beroendena mellan delade resurser i minnessubsystemet och arbetslasters prestanda. Därefter studeras mekanismer för att styra dessa resurser i syfte att motverka de negativa effekterna av överbelastning. Med denna bakgrund introducerar avhandlingen ett nytt ramverk för observerbarhet som kvantifierar resursöverbelastningens påverkan på arbetslasters prestanda under körning och tillhandahåller den information som krävs för att resonera kring resursbeteende på arbetslastnivå. Slutligen sammanförs dessa insikter i en överbelastningsmedveten och intentionsdriven ansats till resurshantering som beaktar både den underliggande infrastrukturens tillstånd och olika intressenters behov. Avhandlingen avslutas med en diskussion om hur dessa principer kan vägleda resurshantering i allt mer heterogena molnbaserade datormiljöer samt om de utmaningar som kvarstår för nästa generation av distribuerade datorsystem.

Preface

This thesis aims to improve workload performance and cluster utilization in cloud computing environments by addressing the challenges posed by shared hardware-level resource congestion. To this end, the thesis comprises a series of research papers investigating different aspects of resource management in cloud native systems. The papers included in this thesis are listed below:

Paper I **O. Larsson**, C. Klein, and E. Elmroth, "The Impact of Directed Pod Eviction on Kubernetes Resource Utilization," in *Proceedings of the 2023 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, pp. 81–90, 2023.

Paper II **O. Larsson**, T. Metsch, C. Klein, and E. Elmroth, "Hardware-Level QoS Enforcement Features: Technologies, Use Cases, and Research Challenges," *ACM Computing Surveys*, vol. 58, no. 6, p. 145:1-145:35, 2025.

Paper III **O. Larsson**, T. Metsch, C. Klein, and E. Elmroth, "ESTHER: Application-First Hardware-Level QoS-Enforcement for Cloud Native Environments," in *Proceedings of the 2025 IEEE International Conference on Cloud Computing (CLOUD)*, pp. 75–85, 2025.

Paper IV **O. Larsson**, T. Metsch, C. Klein, and E. Elmroth, "Workload Buoyancy: Keeping Apps Afloat by Identifying Shared Resource Bottlenecks," *Submitted for publication*, Umeå University, 2026.

Paper V **O. Larsson**, T. Metsch, C. Klein, and E. Elmroth, "Beyond the Limits: Flexible and Congestion-Aware Cluster Scheduling for the Cloud," in *Proceedings of the 2026 IEEE International Conference on Cloud Engineering (IC2E)*, 2026.

Paper VI **O. Larsson**, T. Metsch, C. Klein, and E. Elmroth, "Holistic Buoyancy-Driven Cloud Native Resource Management," *Submitted for publication*, Umeå University, 2026.

Paper VII **O. Larsson**, T. Metsch, and E. Elmroth, "In the Face of Another Generational Shift in IT Infrastructure," *Submitted for publication*, Umeå University, 2026.

In addition to the papers included in this thesis, the following papers were contributed to within the context of the studies.

Paper VIII C. Nguyen, E. Seo, M. Zahid, **O. Larsson**, F. T. Pokorny, and E. Elmroth, "tinyKube: A Middleware for Dynamic Resource Management in Cloud-Edge Platforms for Large-Scale Cloud Robotics," in *Proceedings of the 2025 IEEE Network Operations and Management Symposium (NOMS)*, pp. 1–8. 2025.

Paper IX N. Rasouli, **O. Larsson**, T. Metsch, and E. Elmroth, "A Cross-layer Taxonomy and Resilience Framework for Edge Computing," *Manuscript*, Umeå University, 2026.

Funding Acknowledgment

Funding for this research was provided in part by the Knut and Allice Wallenberg foundation under grant KAW 2019.0352 and by the eSSENCE Programme under the Swedish Government's Strategic Research Initiative. Some works were supported with compute resources provided by the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations.

Declaration of Generative AI Use

Generative AI (OpenAI GPT-5.6) was used to improve the language of the thesis kappa. Some included papers (**Paper IV**, **Paper V**, **Paper VI**, **Paper VII**) were similarly improved with the help of generative AI. The tools and the extent of their use is explicitly acknowledged in the respective papers. The rest of the included papers (**Paper I**, **Paper II**, **Paper III**) were written without any use of generative AI. The author takes full responsibility for the content, accuracy, and scientific analysis of the entire thesis.

Acknowledgments

Throughout the journey of my PhD studies, I have been fortunate to receive support from numerous individuals who have contributed to both my professional and personal growth. I am forever grateful for all the guidance, encouragement, and inspiration, and for the friends I have made along the way. Whether named or not, all of you have played a role in shaping my path, and I extend my heartfelt thanks to each and every one of you.

To start from the beginning, I would like to thank **Paul** for inspiring me to dive deeper into the field of distributed systems and opening my eyes to research. Without taking your courses during the fall semester of 2021, I would never have embarked on this journey. Another key component to me deciding to pursue this PhD was **Simon**. You starting your own PhD studies in the same field, a little over half a year before I would finish my master's degree, allowed me to get insight into what pursuing a PhD entails. Before this, I had no reference point whatsoever, and ultimately, you made me realize that even someone like me could aspire to complete one. Then you gave me an existential crisis by quitting a week after I started. Jerk.

Jokes aside, I cannot write this section without acknowledging the guiding star that has been my supervisor, **Erik**. Thank you for your never-ending support throughout this journey. I have learned so much from you about research, the scientific community, and collaboration. You have always supported my, sometimes rather dumb, ideas and let me explore my own path. I am grateful for your mentorship and for the opportunities you have provided me with, and for occasionally helping me say no to things I would otherwise have regretted saying yes to. You provided me with an environment where I could develop as a researcher, as a professional, and as a person. Thank you for believing in me from the start.

Another key figure has been my co-supervisor, **Cristian**. It is a privilege to have had the opportunity to work with you over the years. Ever since you supervised and guided my master's thesis all those years ago, you have been key to my development as a researcher. All the technical discussions, the brainstorming sessions, feedback, and the occasional life advice you have provided me with have been invaluable. You and your endless optimism are truly inspirational. You lift the people around you to heights they did not think they could reach.

While not an official supervisor on paper, **Thijs** has been a supervisor in all but name and a great source of support and guidance. Your experience and your willingness to share your knowledge and ideas have been invaluable to me. You made me change my mindset and approach to research, and I am grateful for the time and effort you have invested in me.

I would also like to thank all my past and present colleagues at the Autonomous Distributed Systems Lab (ADSLab) for their support and encouragement. I have learned a lot from working with you, and I am grateful for the perspectives and insights you have shared with me. Thank you, **Paul**, for all the banter that livened up otherwise rough days. Thank you, **P-O**, for all the stories, advice, and the late-evening walks back home from *Lottas Krog & Pub*. Thanks to the other PhD students who started at the same time as me: **Obaidullah**, **Pim**, and **Tanaz**. Even though our paths have diverged, I am grateful for the time we spent together and the support we provided each other. Thanks, **Tobias** and **Mohammad Reza**, for the insightful discussions and for the early inspiration to finish this thing by showing me that it is actually possible. Thanks also to **Monowar**, **Johan**, **Chanh**, **Jakub**, **Ali**, **Javad**, **Lidia**, **Charlie**, **Alexandra**, **Maryam**, **Clément**, **Yashwant**, **Antonio**, **Anindya**, **Adil**, **Angela**, **Yinuo**, **Yangyang**, **Ileet**, and **Rohail**, for the great discussions and for making the lab such an inspiring place to work.

I would also like to thank the key individuals at the Department of Computing Science for their support and their guidance. Thanks, **Tomas** and **Mattias**, for all the technical support. It is truly inspirational to see how you can solve any problem that comes your way. Thank you, **Jonny** and **Jerry**, for your encouragement and support during my teaching assignments. Thank you, **Heini**, **Ennie**, **Lili**, **Kai-Florian**, and **Anne-Lie**, for helping me with all the problems and questions I had during my time at the department. Thanks to **Igor**, **Arvid**, **Kaan**, and everyone else who has contributed to making the department a great place to work and study.

During the course of this PhD, I had the privilege of spending a lot of time at the Royal Swedish Academy of Engineering Sciences (IVA). This experience has been invaluable, and I am forever grateful for the opportunity to work with such a talented and dedicated group of individuals. Thank you, **Maria**, **Joakim**, and **Sylvia**, among so many others, for allowing me to grow in ways I never thought possible. I cannot mention IVA without thanking my fellow former members of the Student Council. Thank you, **Benjamin**, for the amazing collaboration we had during our time as chair and vice-chair of the council. I learned more from our collaboration than you know. Also, thank you, **Lukas**, **Diana**, **Maja**, **Eric**, and **Teo**. It has been an honor to work alongside you all.

My family has been a constant source of support and encouragement throughout this journey. I am grateful for their unwavering belief in me and for always being there to listen, offer advice, and provide a shoulder to lean on. **Mom**, **Dad**, and my siblings, **Ludvig** and **Cornelia**, have always been there for me.

Finally, **Matilda**, my partner and fiancée, thank you for your unwavering support and encouragement throughout this journey. You have been my rock

and source of strength during the most challenging times. Thank you for believing in me, even when I did not believe in myself. I would never have done this without you by my side, and I am forever grateful for your love and support.

Oliver Larsson

Umeå, October 2026

Contents

1 Introduction **1**
- 1.1 Cloud Computing . . . 2
  - 1.1.1 Kubernetes . . . 3
  - 1.1.2 Public Clouds . . . 4
  - 1.1.3 Overcommitment and Noisy Neighbors . . . 5
  - 1.1.4 Conflicting Interests of Stakeholders . . . 5
- 1.2 Research Objectives . . . 6
- 1.3 Thesis Outline . . . 8

2 The Lifecycle of Workloads **9**
- 2.1 Latency-Critical Services . . . 12
  - 2.1.1 Creation and Scheduling . . . 12
  - 2.1.2 Scaling Mechanisms . . . 14
  - 2.1.3 Monitoring . . . 15
- 2.2 Best-Effort Jobs . . . 16

3 Shared Hardware-Level Resource Congestion **17**
- 3.1 Shared Hardware Resources . . . 17
- 3.2 Control Mechanisms . . . 18
  - 3.2.1 The Linux User Interface for Resource Control . . . 19
  - 3.2.2 LLC Allocation . . . 20
  - 3.2.3 Memory Bandwidth Limiting . . . 21
- 3.3 Managing Congestion in the Cloud Is Hard . . . 22
  - 3.3.1 Indirectness of Control Mechanisms . . . 23
  - 3.3.2 Lack of Visibility . . . 23

4 Intent-Driven Resource Management **25**
- 4.1 Intent in Cloud Resource Management . . . 26
- 4.2 The Potential of Intent-Driven Resource Management . . . 27

5 Research Methods **29**

6 Thesis Contributions **33**
6.1 Paper I 37
6.2 Paper II 38
6.3 Paper III 39
6.4 Paper IV 40
6.5 Paper V 41
6.6 Paper VI 42
6.7 Paper VII 43

7 Related Work **45**
7.1 Cloud Resource Management 45
7.1.1 Application Owner Perspective 45
7.1.2 Resource Owner Perspective 46
7.2 Hardware-Level Resource Management 46
7.2.1 Black-Box Approaches 47
7.2.2 Application-Aware Approaches 48
7.3 Intent-Driven Resource Management 49
7.4 Summary and Research Gap 50

8 Conclusions **53**
8.1 Future Outlook 54

Bibliography **55**

Chapter 1

Introduction

Technology and software have become integral to almost every aspect of modern society. They shape how we communicate, work, access essential public services, and interact with the world around us. What was once a tool primarily for automating calculations and processing information has evolved into an essential component of everyday life.

Communication provides a particularly visible example of this transformation. Social media platforms have become a primary means of social interaction and entertainment [5], while smartphones provide near-constant access to these services. Financial services have undergone a similar transformation. In many parts of the world, such as in Sweden, physical currency has been almost completely replaced by digital payments [6], making everyday economic activity entirely dependent on digital banking infrastructure. Activities that previously required a transaction of physical currency now depend entirely on the availability of software and the infrastructure supporting it.

This dependence is equally evident in the workplace. Across a wide range of occupations, digital technologies have transformed how work is performed and coordinated. In white-collar professions such as banking, engineering, and administration, digitization has contributed to substantial gains in individual worker productivity. These gains stem in part from enabling workers to manage multiple tasks more effectively, allowing them to quickly switch between tasks when blocked or interrupted [7]. Communication and collaboration tools have further reduced the constraints imposed by geographical distance, making it easier for distributed workers to coordinate their efforts.

The role of software, however, is increasingly extending beyond supporting human activities. Advances in sensing, automation, and machine learning are bringing software into traditionally physical occupations and processes. In agriculture, for example, machine-learning and machine-vision systems can be used to detect crop diseases and pests, enabling farmers to take targeted measures and substantially reduce the use of pesticides [8]. Similar developments are emerging across manufacturing, transportation, healthcare, and other domains,

where software has become an integral part of systems that directly interact with the physical world.

As software becomes embedded in an increasingly broad range of activities, society's dependence on the computing systems that execute this software grows with it. The consequences of a failure are no longer limited to an inaccessible website or an inconveniently slow application. Software increasingly supports services and processes on which individuals, organizations, and society depend. Ensuring that these systems can provide the required performance, reliability, and efficiency is therefore becoming increasingly important.

At the same time, the computing systems supporting this software are becoming increasingly complex. Modern applications rarely execute on a single machine or even within a single administrative domain. Instead, they rely on distributed infrastructures composed of heterogeneous hardware and software components, often shared among many applications and users. Cloud computing and cloud native technologies have further accelerated this trend by allowing computing resources to be dynamically and autonomously provisioned, distributed across large-scale infrastructures, and shared among diverse workloads.

This combination of increasing societal dependence and increasing infrastructure complexity creates new challenges for how computing resources are managed. Applications compete for shared resources, workloads change over time, and the performance experienced by one application can depend on the behavior of others. Understanding and managing these interactions is essential for ensuring that modern computing infrastructures can meet the demands placed upon them.

1.1 Cloud Computing

The term "cloud computing" has in the past two decades become widely known in both technical and non-technical contexts. The wide adoption of everything from cloud storage to cloud-based software-as-a-service (SaaS) applications has made the term familiar to most people, even if they do not fully understand what it means. Before we discuss cloud computing in more detail, it is useful to briefly review its historical context and the factors that have contributed to its popularity.

The client-server software architecture was an important precursor to the modern compute clouds. It experienced rapid growth in popularity in the 1990s [9] due to its ability to reduce the complexity of information flow and coordination, as well as the costs of software development and maintenance.

As computing workloads became more centralized, the benefits of *economies of scale* and the potential for resource sharing became more apparent. As history has shown, centralizing tasks and worker skills can lead to significant efficiency gains, both in terms of cost and performance [10]. This is a principle that has been observed in many domains, from manufacturing to service industries, and

it applies to computing as well.

In 2001, an early prototype of such a system was published under the name "Océano" [11]. It was envisioned as an event-driven system with service-level agreement (SLA) insights that would dynamically allocate resources to workloads based on their needs and priorities, and autonomously recover from failures, not too unlike modern cloud computing platforms. Océano can be considered a precursor to modern cloud computing that, had it succeeded, could have made the industry adopt the term "ocean computing" instead of cloud computing. Alas, this never happened, and the start of practical and commercial cloud computing is often associated with the launch of Amazon Web Services (AWS) in 2006 and the opening of its Simple Storage Service (S3) to the public [12].

Several platforms have been developed to support cloud computing. Some of the most widely used open-source alternatives are OpenStack [13] and Kubernetes [14]. OpenStack is primarily used for managing virtual machines, while Kubernetes is focused on orchestrating containerized workloads. In this thesis, we focus on Kubernetes and its ecosystem, as it has become the de facto standard for orchestrating containerized workloads in cloud environments. However, the concepts, challenges, techniques, and solutions discussed in this thesis are applicable to other cloud computing platforms as well.

1.1.1 Kubernetes

Kubernetes was first introduced in 2014 by Google as an open-source platform for automating the deployment, scaling, and management of containerized workloads [15]. It was built upon the learnings of operating Google's internal cluster management system, Borg [16], which had been in use for over a decade at the time. What made it different was that it leveraged the rapidly growing popularity of Docker [17], the containerization platform that made Linux containers accessible to a wider audience than ever before thanks to new and better abstractions and tooling. Kubernetes has since become the de facto standard for orchestrating containerized workloads and has become the backbone of modern cloud environments, with 82% of organizations reporting that they use Kubernetes in production according to a 2025 survey [18].

The reasons for Kubernetes' success are many, but one of the most important is its ability to abstract away the underlying infrastructure and provide a consistent interface for deploying and managing workloads that is not only portable, but extensible and flexible, allowing users to define their own custom resources and controllers to modify its functionality and behavior to their needs. It takes a configuration-as-data approach, treating all infrastructure as data objects stored in a central database [19]. Users will *declare* the desired state of their system and Kubernetes will continuously work to ensure that the actual state of the system matches the desired state. Initially, this was limited to the deployment and management of containerized workloads, but due to its popularity and flexibility, it has since been extended to manage a wide range

of resources. For example, using the Kubernetes Cluster API [20], users can manage not only containers, but also virtual machines, bare-metal servers, and even entire additional Kubernetes clusters, all from within Kubernetes itself.

1.1.2 Public Clouds

Following AWS's initial lead, several companies have followed suit and taken up the principles of economies of scale and resource sharing by offering their own public cloud platforms. A public cloud is an on-demand computing platform where users rent computing resources from a provider rather than owning and maintaining their own infrastructure. This model offers advantages to both users and providers.

For users, otherwise known as *application owners*, public clouds eliminate much of the responsibility associated with operating physical infrastructure, including managing hardware, power, cooling, and data center facilities. They also shift the cost of computing from large upfront capital expenditures (CapEx) to more manageable monthly operational expenditures (OpEx), providing a more flexible way to acquire computing capacity [21]. Furthermore, users generally pay only for the capacity they consume and can scale their provisioned resources up or down as needed. This flexibility is particularly valuable for workloads with variable demand, such as many user-facing applications.

For providers, also known as *resource owners*, the public cloud model enables economies of scale by consolidating computing resources into large, warehouse-scale data centers. The largest public cloud providers, including AWS, Microsoft Azure, and Google Cloud Platform (GCP), are commonly referred to as hyperscalers. Many hyperscalers also have substantial internal demand for computing resources to support their own products and services [22]. Consequently, they already have strong incentives to build and operate large-scale data center infrastructure. Although building and maintaining such infrastructure is expensive, this investment also provides hyperscalers with an opportunity to leverage their existing infrastructure, tooling, and expertise to serve external customers. By sharing infrastructure between internal and external workloads, providers can spread the costs of facilities, backup power, cooling, and other infrastructure across a larger pool of users. The public cloud model therefore allows hyperscalers to turn infrastructure that would otherwise primarily serve their own operations into a shared platform for both internal and external workloads.

The public clouds have seen immense growth in recent years, with cloud spending nearly double that of on-premises IT spending already in 2021 [23]. Considering the growth trajectory that, had it held up, would have made cloud spending orders of magnitude larger than on-premises IT spending by today, it is clear that the public cloud model has been widely adopted and is likely to continue to play a central role in the computing landscape for the foreseeable future. However, it is not without its challenges.

1.1.3 Overcommitment and Noisy Neighbors

It is widely known that utilization of computing resources in data centers is often low, with most servers operating at only a fraction of their capacity [24], [25]. This can be attributed to several factors, including unsold capacity, resource stranding, and underutilization due to overprovisioning, where customers pay for more resources than they actually use [26].

One of the most common strategies that resource owners leverage to improve resource utilization is *overcommitment*, where the total amount of resources allocated to workloads exceeds the physical capacity of the system [27]. This helps to improve hardware utilization, but it simultaneously introduces additional resource congestion. When multiple workloads compete for the same shared physical resources, the performance of one workload can be affected by the resource usage of others, even if they are logically isolated from each other [28]. This phenomenon is commonly referred to as the "noisy neighbor" effect and appears not only in bare-metal scenarios, but in virtualized and containerized environments as well [29]. We will discuss resource congestion and mitigation mechanisms in more detail in Chapter 3.

With the environmental, social, and economic challenges of operating IT infrastructure that have become much more pressing in recent years, it is paramount that we leverage the computing resources we have in the most efficient way possible. Letting hardware sit underutilized is not only wasteful, but it is also one of the least energy-efficient ways to operate a data center [30]. However, achieving an improvement in resource utilization is not without its challenges. First, there is the challenge of defining what "better" means.

1.1.4 Conflicting Interests of Stakeholders

In the context of compute clouds, there are multiple stakeholders with different interests and objectives. At a high level, we can identify two primary groups of stakeholders: application owners and resource owners.

The application owners are primarily concerned with the performance and reliability of their applications and will provision resources to meet their application requirements. Generally, some overhead is included in the provisioning to account for variations in workload demand, noisy neighbors, and other uncertainties. There may also be a lack of granularity available in the resource allocation mechanisms. This overhead leads to underutilization, which is a problem for the resource owners, who are responsible for managing the underlying infrastructure. Their goal is to maximize the sold capacity and utilization of their resources while ensuring that customer-facing SLAs are met, as violating SLAs can have financial and reputational consequences. To do so, they will overcommit resources and reserve more capacity than is physically available. This causes resource congestion, leading to performance degradation for some applications. In turn, some application owners will increase their resource allocations to compensate for the performance degradation, which further exacerbates

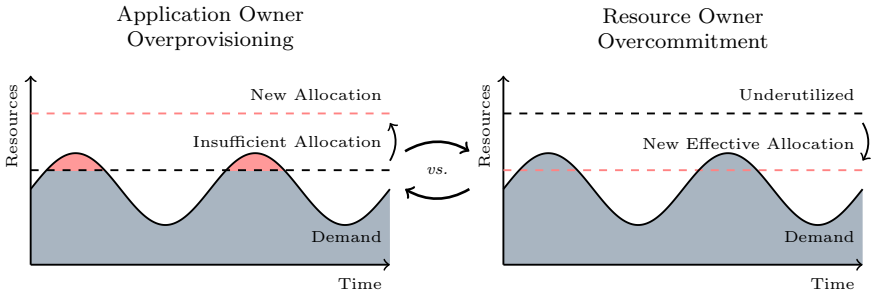


Figure 1.1: The vicious cycle of overprovisioning and underutilization. Application owners overprovision resources to maintain headroom and ensure their demands are met. This creates an underutilization problem for resource owners, who overcommit resources to improve utilization. As more resources are allocated to workloads than are physically available, resource congestion arises, leading to performance degradation for some workloads as the effective allocation decreases. This in turn causes some application owners to increase their resource allocations, creating a vicious cycle.

the problem of underutilization.

This becomes a vicious cycle where the interests of application owners and resource owners are in conflict, as illustrated in Figure 1.1. To improve utilization, focus must be placed on mitigating the underlying resource congestion, which requires a better understanding of the interactions between workloads and the shared resources they compete for. To complicate this, most applications and their application-level metrics are often completely opaque to the resource owners, for technical, security, or even legal reasons. Furthermore, even when application-level insights are available, they rarely map directly to the underlying hardware-level resource control mechanisms, making it difficult to understand how application-level behavior translates to hardware-level resource congestion [3], [31].

1.2 Research Objectives

The overarching goal of this thesis is to improve the effective utilization, performance, and, consequently, the computing efficiency of cloud computing infrastructures by managing and mitigating resource congestion. Here, effective utilization refers to utilizing available computing resources for as many useful computations as possible while maintaining the service-level objectives (SLOs) of the workloads executing on them. Increasing utilization without considering application performance can lead to resource congestion and SLO violations, limiting the amount of useful work that can be performed. By enabling more workloads to meet their SLOs with the same amount of computing hardware,

the energy consumption per unit of useful work can be reduced, thereby contributing to a lower environmental impact of computing. To pursue this goal, three research objectives are defined that are motivated by technical insights and related work discussed in later chapters. They are presented here to provide relevant context for the rest of the thesis.

A fundamental challenge to improving effective utilization is resource congestion, which arises when multiple workloads compete for shared hardware resources. Although mechanisms for controlling hardware-level resources exist, they are often difficult to use effectively in cloud environments because their effects are indirect and their relationship to application-level behavior is not readily apparent [2]. The first research objective therefore focuses on understanding this relationship and establishing mechanisms for connecting application-level behavior with hardware-level resource control:

RO1 Characterize hardware-level resource congestion and develop techniques for relating application-level behavior to hardware-level resource control.

Cloud environments are inherently dynamic, with the composition and resource demands of workloads changing over time. Moreover, the infrastructure typically has limited insight into how individual workloads will behave and interact with other workloads competing for the same resources. These characteristics make it challenging to manage shared resources effectively while maintaining workload SLOs. Building on the understanding developed for **RO1**, the second research objective therefore focuses on resource management at the node level:

RO2 Develop node-level strategies for managing shared hardware resources to improve effective utilization while maintaining workload SLOs.

Finally, effective resource management must consider not only individual nodes but also the cluster as a whole. Decisions made at the node level can affect how resources are distributed across the cluster and may result in fragmentation, where available resources cannot be effectively utilized by incoming workloads. The third research objective therefore extends the scope of the previous two objectives to cluster-level resource management, with the aim of improving resource utilization while maintaining the SLOs of workloads:

RO3 Develop cluster-level resource management strategies that improve effective resource utilization by mitigating resource fragmentation while balancing the needs of application owners and resource owners.

By addressing these research objectives, this thesis strives to improve the effective utilization of cloud computing infrastructures through vertical integration. By leveraging insights and control mechanisms at both hardware and cluster levels in a coordinated manner, it is possible to achieve better resource utilization and efficiency with minimally invasive changes to existing workloads and infrastructure.

1.3 Thesis Outline

The remainder of this thesis is organized as follows. Chapter 2 provides an overview of the workload lifecycle in cloud computing environments, including workload characteristics, scheduling, and resource management. Chapter 3 focuses on hardware-level resource congestion and techniques for mitigating its effects. Chapter 4 explores the notion of intent and how it can be used to convey what applications need from the underlying infrastructure. In Chapter 5, we discuss the research methods used in this thesis. Chapter 6 presents the main findings and contributions, along with the included papers and how they relate to the research objectives. Finally, Chapter 7 discusses related work, while Chapter 8 summarizes the key findings of the thesis and outlines directions for future research.

Chapter 2

The Lifecycle of Workloads

When discussing computing workloads, their behavior, and their requirements, it is important to understand the different stages of their lifecycle. How is it decided that a workload should be created? What resources does it require? Which hardware should it run on? Once started, is it running as expected? Are its resources sufficient, or perhaps excessive? If it stops, is it because of a failure, or because it has completed its task? Answering these questions, among many others, is part of managing computing workloads throughout their lifecycle.

Crucially, the answers to these questions may differ depending on the type of workload. For the most part, workloads can be divided into two main categories: *latency-critical* (LC) services and *best-effort* (BE) jobs. This distinction applies to workloads hosted in the cloud, on bare-metal servers, and even on personal computers.

Latency-critical services are long-running workloads that are expected to respond to incoming requests in a timely manner. Without incoming requests, they may not perform much work, but they are expected to be ready to respond at any time. Examples of latency-critical workloads include web servers, database systems, and DNS (domain name system) servers. Such workloads generally do not terminate on their own. When they stop, it is usually because of a failure or an administrator-initiated shutdown. Upon failure, latency-critical workloads can be configured to restart automatically and resume operation as quickly as possible.

When operating latency-critical workloads, application owners are often concerned with service-level indicators (SLIs) that quantify the quality of service (QoS) provided to users. Such indicators include, but are not limited to:

- **Latency**—The time it takes to respond to a request, commonly characterized as a tail latency percentile (e.g., 95^{th} or 99^{th}). Depending on the workload, latency may be measured at different timescales, but milliseconds or seconds are common resolutions.

- **Throughput**—The number of requests that can be handled per unit of

time. Throughput is often measured in requests per second (RPS) or transactions per second (TPS).

- **Jitter**—The variation in latency between requests, often measured as the standard deviation of request latencies.

- **Availability**—The fraction of time the service is operational and able to handle requests.

- **Error Rate**—The fraction of requests that fail to be successfully handled by the service.

The most important SLI for a given workload depends on its purpose and the expectations of its users. For example, a web server for an online retail store may prioritize low latency, as slow website response times have been shown to negatively impact sales [32]. Conversely, an internal data processing service may prioritize throughput to ensure that large amounts of data are processed in a timely manner, even if this means that individual requests take slightly longer to complete.

The other major type of workload found in computing environments is the best-effort job. A best-effort workload is a short-lived task that is expected to complete and terminate once its work is finished. Here, "short-lived" is used as a relative term with respect to the theoretically infinite lifetime of latency-critical workloads. Examples of best-effort workloads include data analysis tasks, media conversion tasks, machine learning model training, and scientific simulations. Their execution time can range from sub-second tasks to computations that take hours or even days to complete. The key characteristic is that they are expected to complete given sufficient resources and time.

Operating best-effort workloads involves a different set of concerns than operating latency-critical workloads. The SLIs considered important for best-effort workloads are often related to the time required to complete the task, the resources consumed, and the cost of execution. Common SLIs for best-effort workloads include:

- **Completion Time**—The total time taken from the start of the task to its completion. In some cases, this may include a group of tasks that are executed sequentially or in parallel, with completion time measured for the entire group.

- **Resource Utilization**—The amount of CPU, memory, and other resources consumed during execution. It is often possible to trade resource utilization against completion time, as allocating more resources may allow a task to complete faster.

- **Cost**—The monetary cost associated with executing the workload, which is especially relevant in public cloud environments.

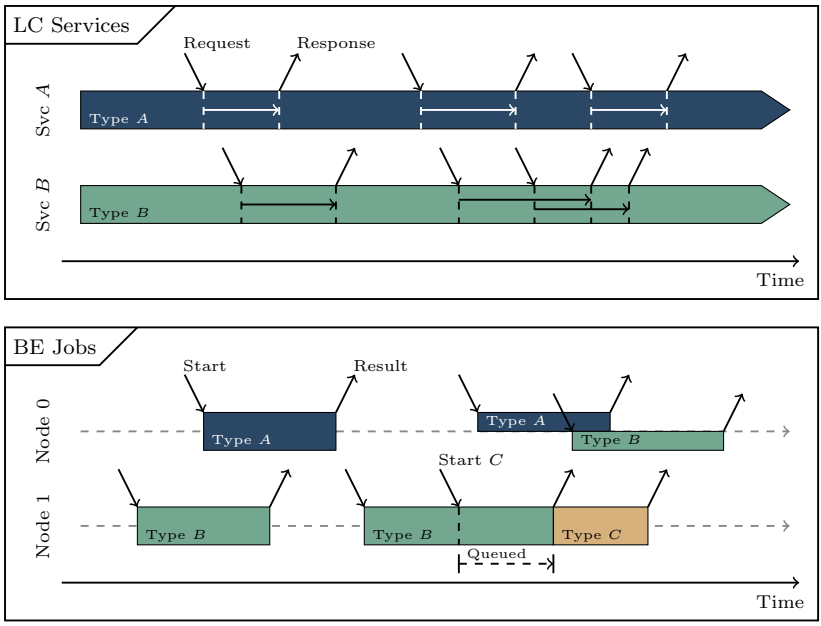


Figure 2.1: Comparison of latency-critical (LC) services and best-effort (BE) jobs. Latency-critical services are continuously running workloads that respond to incoming requests, while best-effort jobs are short-lived tasks that complete and terminate once their work is finished.

- **Error Rate**—The fraction of tasks or sub-tasks that fail to complete successfully.

Similarly to latency-critical workloads, the most important SLIs for best-effort workloads depend on the specific use case and the expectations of the users. For example, a scientific simulation may prioritize completion time to ensure that results are obtained quickly, while a batch data processing job may prioritize resource utilization and cost minimization. An overview of the differences between latency-critical and best-effort workloads is illustrated in Figure 2.1.

It could be argued that each request to a latency-critical service is itself a best-effort task that terminates once the response has been sent. However, the performance targets and SLIs considered important are generally different. In the remainder of this chapter, we primarily focus on the lifecycle of latency-critical services, as these are the workloads most relevant to the contributions of this thesis. Despite this focus, we briefly discuss best-effort jobs and their role in improving cluster utilization in Section 2.2.

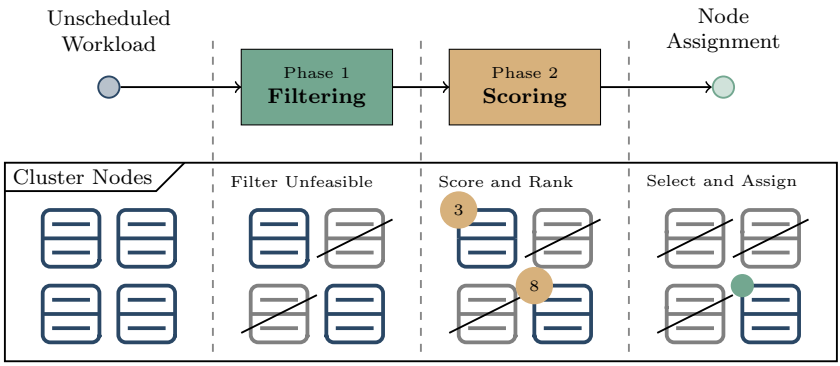


Figure 2.2: An overview of the two-phase scheduling approach used by many cloud orchestration systems.

2.1 Latency-Critical Services

Latency-critical services are fundamental building blocks of commonly used client-server and service-oriented software architectures. In this section, we explore how latency-critical services are created, scheduled, executed, and monitored.

2.1.1 Creation and Scheduling

The first step in a workload's lifecycle is deciding that the workload is needed. This decision may be made either by a human administrator or by an automated system. During this stage, the resource requirements of the workload must also be determined. Such resources include CPU and memory, as well as other resources required for the workload to function properly. There may be storage volumes that need to be mounted, runtime secrets that need to be provided, or special networking requirements that must be satisfied. These requirements must be specified by the workload's creator and are subsequently used by the scheduling system to determine where the workload should be placed.

Scheduling is the process of assigning workloads to available resources in a cloud computing environment. The scheduler is the component responsible for making these decisions based on the workload's requirements and the current state of the system. Its goal is to ensure that workload requirements are met while optimizing cluster resource utilization. Furthermore, schedulers may attempt to balance the load across available resources to improve resilience against failures, such as hardware failures.

Many cloud orchestration systems, including Kubernetes, OpenStack, and Borg, use a *two-phase* scheduling approach when assigning workloads to worker nodes [33]. The two phases are *filtering* and *scoring*. An overview of the process is illustrated in Figure 2.2.

In the first phase, the filtering phase, the scheduler evaluates all available compute nodes and filters out any node that does not meet the workload's requirements. Nodes that are filtered out are considered *infeasible* and are no longer considered during the scoring phase. Filtering criteria can include, but are not limited to, the following:

- **Resource Availability**—The node must have sufficient CPU, memory, and other resources to meet the workload's requirements.

- **Affinity Requirements**—The workload may have specific requirements regarding the type of node on which it can run, such as hardware architecture or geographic location.

- **Taints**—Nodes may have rules applied to them that control which workloads may be scheduled on them. Such rules, referred to as *taints* in Kubernetes, allow operators to specify conditions under which workloads can be scheduled on a node. For example, some nodes may be designated for specific control plane workloads, such as the scheduler itself. Other nodes may not accept workloads at all, such as nodes that are down for maintenance.

If no node passes the filters and is considered feasible, scheduling has failed. The scheduler may either reject the workload entirely or, more commonly, attempt to schedule it again at a later time.

After the filtering phase, the scheduler enters the scoring phase. In this phase, all feasible nodes are assigned a score based on how well they can accommodate the workload's requirements given the current state of the system. The scores are used to rank the nodes from most to least suitable for the workload. The node with the highest score is selected as the target node for the workload. If multiple nodes tie for the highest score, the scheduler uses a tie-breaking mechanism to ensure that a single node is selected. In many cases, the tie-breaking mechanism can simply be random selection among the nodes that share the highest score.

Scoring criteria can include, but are not limited to, the following:

- **Resource Utilization**—Nodes with lower resource utilization may be preferred to ensure that workloads are distributed evenly across the cluster.

- **Data Locality**—If the workload requires access to specific data, the scheduler may prefer nodes that are closer to the data source to reduce latency.

- **Replica Spread**—For workloads with multiple replicas, the scheduler may prefer nodes that are not already hosting a replica of the workload to ensure that replicas are distributed across different nodes for fault tolerance.

Many schedulers allow custom sets of filtering and scoring functions to be defined, altering the behavior of the scheduler without requiring modifications to the scheduler itself. This allows operators to customize the scheduling process and implement their own policies and requirements. For example, a custom scoring function could incorporate information about which nodes are experiencing application-level performance issues. The function could assign lower scores to such nodes to avoid scheduling additional workloads on them and thereby further increasing congestion. Such custom scheduling policies and plugins are leveraged in **Paper I**, **Paper V**, and **Paper VI**.

Once the workload has been assigned to a node, a node-level component is responsible for starting the workload and ensuring that it remains operational. In Kubernetes, the *kubelet* communicates with the container runtime to start the workload's container and monitor its health. In OpenStack, the *compute service* is responsible for starting the workload's virtual machine.

2.1.2 Scaling Mechanisms

A common requirement for latency-critical workloads is to have the ability to scale up or down in response to changes in incoming load. Demand may follow diurnal or seasonal patterns, or may change abruptly due to sudden spikes in demand [24], [34].

To handle such variations without continuously overprovisioning resources, latency-critical workloads can be designed to scale up during periods of high demand and scale down during periods of low demand. This ensures that only the necessary resources are used at any given time, potentially reducing costs and improving utilization of the underlying infrastructure. Scaling can be performed in two ways:

- **Vertical Scaling**—In vertical scaling, the resources allocated to a single workload instance are increased or decreased. This can be done by adjusting the CPU and memory limits for the workload. Vertical scaling is often used for stateful workloads, where increasing the resources of a single instance can improve its performance. However, vertical scaling is limited by the resources available on a single node and may require the workload to be restarted for the new resource limits to take effect. It is analogous to buying a faster computer when the current one is no longer fast enough to keep up with demand.

- **Horizontal Scaling**—In horizontal scaling, the number of workload instances is increased or decreased. This is achieved by adding or removing replicas of the workload, which can be distributed across multiple nodes. Horizontal scaling is often used for stateless workloads, where each instance can handle requests independently. Applications built for cloud environments are generally designed with horizontal scaling in mind, as it enables better fault tolerance and load distribution.

Figure 2.3 illustrates the difference between vertical and horizontal scaling.

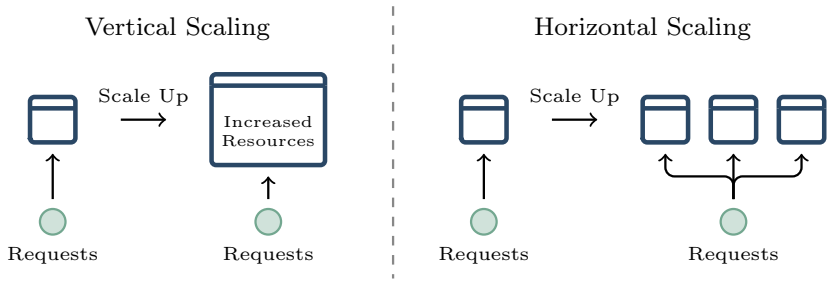


Figure 2.3: Illustration of vertical and horizontal scaling mechanisms. When scaling vertically, the resources allocated to a single workload instance are increased or decreased. When scaling horizontally, the number of workload instances is increased or decreased.

2.1.3 Monitoring

Monitoring sandboxed workloads, such as containerized or virtualized workloads, can be performed at three levels. First, the workload itself can export metrics and health information at the application level. Such metrics may include application-specific information, such as request latency or error rates. Second, at the container or virtual machine level, the runtime can export metrics about the state of the workload process to determine whether it is still running or has exited and, if so, whether it exited successfully or with an error. The runtime can also export resource usage metrics, such as CPU and memory utilization. Third, at the node level, the operating system can export hardware-level counters and metrics that can be used to monitor workload performance with respect to resource usage and congestion.

Commonly, only application-level and runtime-level metrics are used to monitor and manage workloads in cloud systems. However, leveraging node-level information about resource usage and congestion to allocate and partition resources accordingly can greatly improve workload performance [2], [35], [36]. In **Paper II**, we review the state of the art in how such node-level insights and control can be leveraged to improve workload performance in cloud systems. One of the key findings of that study is that the lack of integration of node-level insights into cloud workload orchestration is partly due to the fact that acting on such metrics and counters can be indirect and non-intuitive [31].

For example, changing the amount of last-level cache available to a workload can have unintended consequences for overall system performance. When the amount of cache available to one workload is increased, other co-located workloads may experience increased cache misses and consequently generate more memory bus traffic. This can indirectly affect the performance of the workload that was allocated additional cache [37]. We discuss node-level resource monitoring, insights, and control in more detail in Chapter 3.

The challenges associated with incorporating node-level insights into work-

load orchestration mean that such insights are often left unused in practice. To address the problems associated with indirect and non-intuitive node-level resource information in the cloud, we propose in **Paper IV** a new approach to monitoring and observability, called *workload buoyancy*, that quantifies the impact of node-level resources on workload performance in a more direct and intuitive manner [38]. We subsequently leverage this approach in **Paper V** and **Paper VI** for both workload scheduling and node-local resource allocation.

2.2 Best-Effort Jobs

While latency-critical services are the primary workloads in many cloud environments, best-effort jobs are also important for improving cluster utilization. Their nature as short-lived tasks that do not always require immediate completion allows them to be scheduled more flexibly. When best-effort jobs are queued to run opportunistically as resources become available, they can fill resource gaps left by latency-critical workloads, using resources that would otherwise be left unused. Thus, during periods of low demand for latency-critical workloads, such as at night, best-effort jobs can be scheduled to utilize otherwise idle resources. When latency-critical load increases, best-effort jobs can be preempted or delayed, ensuring that latency-critical workloads have the resources they need to meet their SLOs. This strategy has been used to great effect by Google in its Borg cluster management system [39], as well as in several resource management research proposals (e.g., [35], [40], [41]).

The key to making such a strategy work is to establish priorities between workloads. The simplest approach is to consider latency-critical workloads to have higher priority than best-effort workloads. This has been shown to be effective in practice [25], [42] and makes sense in cases where best-effort workloads are used for non-critical tasks such as data analysis or media conversion. We use this prioritization in **Paper III** to ensure that latency-critical workloads are not adversely affected by co-located workloads, while allowing those other workloads to use as much of the remaining resources as possible [3].

In other cases, more granular prioritization may be needed. For example, some best-effort workloads may be more important than others, requiring a more nuanced approach to scheduling and resource allocation, such as the one used by Borg [39].

Chapter 3

Shared Hardware-Level Resource Congestion

To be able to understand why multitenancy in cloud environments causes resource congestion, and the challenges of managing it, we first need to understand what shared hardware resources are and what control mechanisms are available to manage them.

3.1 Shared Hardware Resources

The multitenant nature of cloud computing environments means that workloads are required to share hardware resources. By being hosted on the same physical machine, workloads must share a variety of hardware resources and components such as CPU cores, memory, and I/O devices. This sharing of resources can lead to congestion, where the applications compete for the limited resources available on the system. This can lead to performance degradation for workloads if their combined demand for a resource exceeds the available capacity.

For some hardware resources, isolation mechanisms have existed for a long time that allow individual workloads to be isolated from each other. For example, CPU cores can be exclusively allocated to a workload, and memory is allocated in a way that prevents workloads from interfering with each other through kernel-level security measures such as virtual address spaces. However, there are many hardware resources that must be shared between workloads. Such resources include CPU caches and other uncore resources,[1] memory bus bandwidth, and I/O devices. These resources can become sources of congestion in shared environments, which can lead to performance degradation for workloads. These resources can generally be classified into one of four categories:

[1] Uncore resources are hardware components that are part of the CPU package but are not part of the CPU cores themselves. Examples include the memory controllers and interconnects [43].

- **Bandwidth Resources**—Bandwidth resources are resources that have a limited capacity for data transfer. Examples include memory bus bandwidth and network bandwidth.
- **Computation Resources**—Computation resources are resources that are used to perform computations on data. Examples include graphics processing units (GPUs) and specialized accelerators such as tensor processing units (TPUs).
- **Cache Resources**—Caches are small and fast memory resources that are used to store frequently accessed data. Examples include CPU caches and storage caches.
- **Storage Resources**—Storage resources are resources that are used to store data. Examples include system memory, solid-state drives (SSDs), and network-attached storage (NAS) devices.

This division is not perfect. However, it is useful for understanding the different types of resources that can become congested in shared environments. One of the key differences between the different types is the way their utilization is measured. Bandwidth resources are typically measured in terms of throughput, whereas computation resources are typically measured in terms of a fraction of cycles used for computation. Cache resources tend to be measured in terms of hit rates and miss rates, which are the fraction of accesses that are served from or miss the cache, respectively, while storage resources are measured in terms of their used versus total capacity.

In most modern cloud environments, workloads can be allocated CPU cores, memory, and sometimes GPU accelerators, while other shared resources are not considered or available to request. As different workloads will be sensitive to different resources, this can lead to performance degradation and subsequent overprovisioning if they are sensitive to resources that are not included in the traditional allocation process [38]. As an example, Figure 3.1 shows how limiting the available last-level cache (LLC) capacity for a workload can cause performance degradation similar to that caused by reducing the CPU core allocation. Even so, allocating a specific LLC capacity to a workload is not possible in most cloud environments, and the workload must be overprovisioned to avoid performance degradation. Luckily, however, modern hardware provides mechanisms for controlling the allocation of some shared resources such as LLC and memory bandwidth.

3.2 Control Mechanisms

Control mechanisms are available for managing the allocation of many shared hardware resources. Most of these mechanisms are implemented in hardware and can be controlled by the operating system or hypervisor. To keep this section concise, we will only discuss the control mechanisms that are relevant

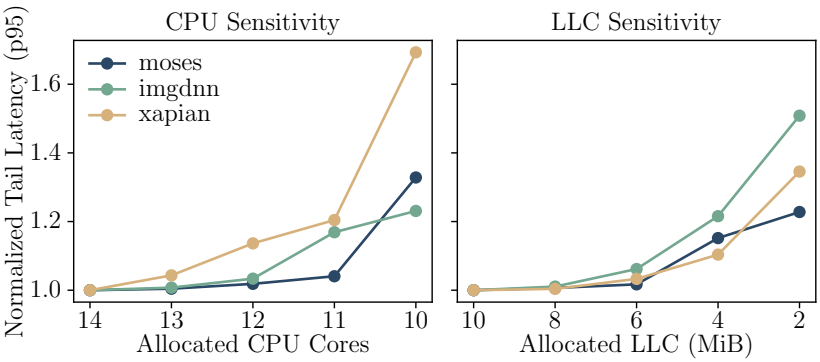


Figure 3.1: Exemplification of how limited shared hardware resources can lead to performance degradation for workloads. Here, we see that limiting available LLC capacity for a workload can cause performance degradation for latency-critical services similar to that caused by reducing the CPU core allocation. This figure is from **Paper IV** [38].

for the rest of this thesis. Thus, we will focus on the control mechanisms that exist within the Linux user interface for resource control [44], specifically for LLC partitioning and memory bandwidth limiting, due to their central roles in **Paper II**, **Paper III**, **Paper IV**, and **Paper VI**.

3.2.1 The Linux User Interface for Resource Control

The Linux user interface for resource control (`resctrl`) is a kernel-level interface for the control of some shared CPU package resources. It provides a largely standardized interface that enables portability across hardware platforms, even if the actual behavior between those platforms may drastically differ [2], [31]. Intel calls this set of features the Intel Resource Director Technology (RDT) [45], while AMD calls it the AMD Platform Quality of Service (PQoS) [46] extensions. Due to their hardware-level specifics, some differences exist between the two implementations.

A core concept of the `resctrl` interface is the notion of *classes of service* (COSs). A COS defines a resource allocation policy that determines how shared hardware resources are allocated to the workloads belonging to that class. Multiple workloads can be assigned to the same COS, in which case they are subject to the same resource allocation policy (see Figure 3.2). For example, a COS can be configured to allocate a particular share of a resource, which is then shared among all workloads assigned to that COS. Since COSs are implemented in hardware, the number of available COSs is limited and depends on the underlying platform [31]. Workloads can be assigned to a COS either through their physical or logical CPU cores or by specifying individual process

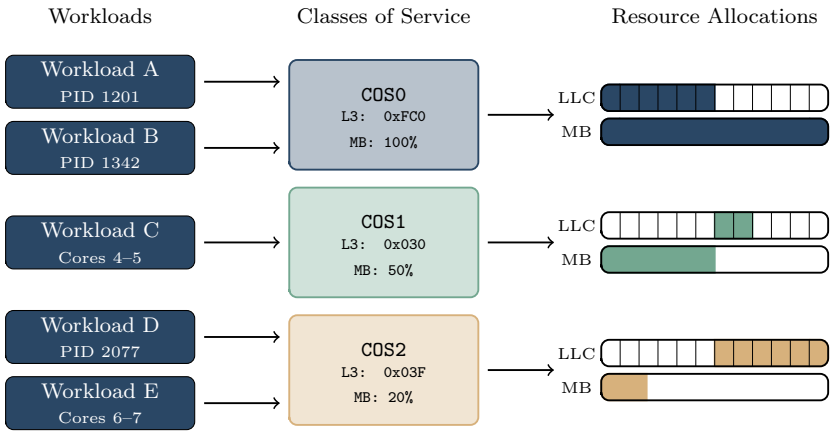


Figure 3.2: Illustration of how workloads can be assigned to classes of service (COSs), where each COS defines a resource allocation policy that determines how shared hardware resources are allocated to the workloads belonging to that class. Multiple workloads can be assigned to the same COS, in which case they are subject to the same resource allocation policy.

IDs.

3.2.2 LLC Allocation

Management and partitioning of shared LLC has a long history in the research community. Several software techniques have been proposed that enable the management of LLC allocation [47], where one of the most prominent ones is page coloring [48].

The notion of QoS in the context of cache allocation was originally introduced by Iyer in 2004 [49]. This concept was further developed, and Intel was first to introduce a hardware-level mechanism for LLC allocation in the Xeon E5-2600 v3 family of processors in 2014 [50]. AMD has since followed suit, and similar efforts have been made for ARM and RISC-V platforms [2].

LLC allocation operates on a way-by-way basis, where each individual cache way may be allocated to a specific COS. The allocation scheme is controlled by an allocation bitmask, where each bit represents a cache way. Cache ways may be exclusively allocated to a COS, or shared between multiple COSs. On some platforms, the '1' bits must be set in a contiguous block [44]. The number of cache ways available and the size of an individual way are platform-specific, and the platform thus affects the granularity of the allocation scheme. An illustration of how the LLC allocation bitmask works is shown in Figure 3.3.

Changes to the allocation scheme only affect new cache allocations. Existing cache lines remain accessible to their owning workloads, even if they reside in cache ways that are no longer allocated to the workloads' COS, until they are

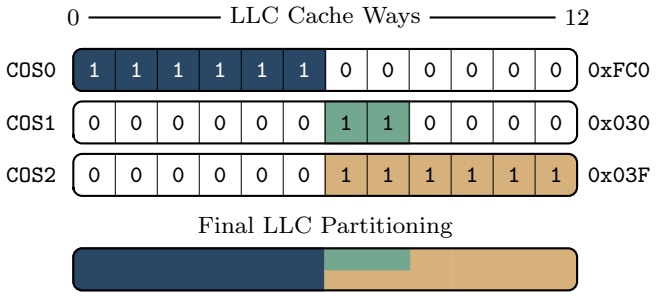


Figure 3.3: Illustration of how LLC allocation works on a way-by-way basis, where each individual cache way may be allocated to a specific COS. The allocation scheme is controlled by an allocation bitmask, where each bit represents a cache way. This figure is from **Paper III** [3].

evicted from the cache. Consequently, the effects of changes to the allocation scheme are not immediate but gradually propagate through the system as existing cache lines are evicted and replaced. The time required for this process depends on the workloads' memory access patterns, meaning that the effects of a change may not become immediately visible [44].

3.2.3 Memory Bandwidth Limiting

Similar to LLC allocation, memory bandwidth management has a long history in the research community, with software-based approaches such as MemGuard [51] having been studied for many years. Hardware-based mechanisms for controlling memory bandwidth have since been introduced by both Intel and AMD and are exposed through the `resctrl` interface. These mechanisms follow the same COS-based model as LLC allocation, where each COS is assigned a maximum memory bandwidth limit. Intel refers to its implementation as memory bandwidth allocation (MBA) [31]. However, as we will see, memory bandwidth limiting (MBL) is a more accurate description of the functionality provided.

MBL works by limiting the memory bandwidth that workloads assigned to a COS can consume. The limits can be configured either as a percentage-based throttling parameter or, on supported platforms, as an absolute throughput limit expressed in MiB/s. In the latter case, a feedback control loop monitors the memory bandwidth consumption of each COS and dynamically adjusts the percentage-based throttling parameter to keep the observed throughput within the configured limit [44].

The limits are typically enforced on a thread-by-thread basis by throttling memory requests before they reach the shared LLC [31]. On some platforms, however, enforcement is performed on a core-by-core basis, meaning that all threads executing on a given core share the same limit. In these cases, the

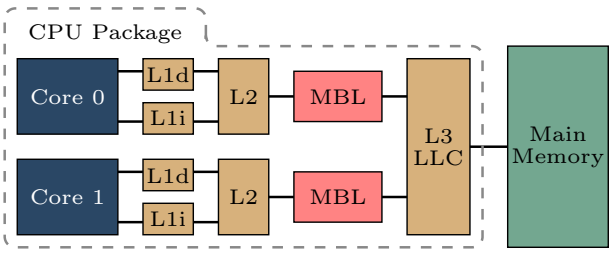


Figure 3.4: Illustration of the memory bandwidth limiting mechanism, where limits are enforced on a thread-by-thread basis by throttling requests before they reach the shared LLC. This figure is from **Paper III** [3].

most restrictive limit among the threads determines the effective limit applied to the entire core [44]. An illustration of the MBL mechanism is provided in Figure 3.4.

In **Paper II**, we investigate how these hardware-level control mechanisms can be leveraged to mitigate resource congestion. The study shows that, although effective, these mechanisms remain largely underutilized in cloud environments. In the following, we examine some of the reasons behind this limited adoption.

3.3 Managing Congestion in the Cloud Is Hard

Controlling the memory subsystem is a complex task that tends to be limited to very specific use cases [2]. Environments such as real-time systems tend to be more amenable to such control, as workload composition and memory access patterns are known in advance. There, manual tuning to an optimized static allocation scheme can be done to ensure that workloads are minimally affected by resource congestion.

Cloud environments, on the other hand, are considerably more dynamic and unpredictable. The composition of workloads on a compute node can change over time, while individual workloads may exhibit changing memory access patterns as they evolve through software updates and changes in operating conditions. Furthermore, security and privacy concerns may limit the cloud provider's ability to inspect and profile customer workloads. These characteristics make it difficult to apply the same assumptions commonly made in static and real-time systems, posing significant challenges for the management of shared hardware resources in cloud environments.

The reasons for this are numerous, but we focus on two main challenges: the indirectness of hardware-level control mechanisms and the lack of visibility, both into workloads and into hardware resources.

3.3.1 Indirectness of Control Mechanisms

The indirectness of the control mechanisms fundamentally stems from their hardware implementation specifics and from the fact that none of the systems exist in isolation. In the case of MBL, the only way to improve the memory bandwidth "allocation" of a workload is to apply limits to other workloads and thus free up more bandwidth to be consumed by the intended workload. Furthermore, since the underlying control mechanism operates on a thread-by-thread (or core-by-core) basis, while the memory bandwidth is shared between all requests on the NUMA node, the actual result of an applied limit is dependent on the memory access patterns of all other workloads that share the NUMA node. This shows up as a non-intuitive and non-linear relationship between the applied limits and the actual memory bandwidth usage of a workload [52].

In the case of LLC allocation, intuition might suggest that allocating more cache ways to a workload should improve its performance. In many cases it does. However, the relationship between the amount of allocated cache and the performance of a workload is not linear and depends on the memory access patterns of the workload and its working set size. More cache may not be beneficial if the workload's working set is smaller than the allocated cache, or if the working set is so large that most memory requests miss the cache regardless of the allocation [38]. There are also cases where allocating more cache to a workload may actually degrade its performance. Allocating more cache to a workload can cause other workloads to miss the cache more often, which may lead to increased memory bus congestion and thus lead to longer memory access times when the workload inevitably does miss the cache [37].

3.3.2 Lack of Visibility

The indirectness of the control mechanisms is a problem that some research has partially overcome, even without prior profiling [2], [36], [41]. However, most such approaches assume full visibility into both hardware- and workload-level metrics. This assumption is often not valid in cloud environments, where different stakeholders have different levels of visibility into the system.

The resource owners, who have the most visibility into hardware- and system-level metrics, generally have no visibility into the application-level metrics of the workloads that are running on the system for security and privacy reasons. Workloads are considered unpredictable black boxes, and there is no way of knowing how they are affected by interference from other workloads. This makes it difficult to reason about the effects of control mechanisms on individual workloads.

On the other hand, application owners have full visibility into their own workloads, but no visibility into the hardware- or system-level metrics. As the composition of workloads changes on a node, they may notice changes in their own workload performance due to changes in shared resource pressure from the other workloads, but they generally have no way of pinpointing what is causing

the change.

Furthermore, even if application owners fully understood the dynamics of their own workloads, there does not exist any mechanism for them to communicate this information to the resource owners. When an application owner shops for a virtual machine, there is no way for them to specify what resources their workload is sensitive to, and the resource owner has no way of knowing what types of workloads are running on the system. Application owners likely cannot even make assumptions about the exact hardware platform their workload will be running on, as it may change over time.

An effort was made to make the cache allocation mechanism available to VM guests by passing some of the control to the guest OS in the vCAT project [53]. However, this approach never gained traction in practice as it would simply shift the problem of managing shared hardware resources to the application owners. Due to the indirectness of the control mechanisms, this would require a deep understanding of the hardware and expertise in performance engineering, which is not a reasonable expectation for application owners.

The workload buoyancy framework introduced in **Paper IV** provides a means to quantify the sensitivity of individual workloads to specific hardware resources. The insights gained from workload buoyancy can be used to develop more effective resource allocation schemes, as demonstrated in **Paper VI**, which leverages this framework to create a congestion-aware and intent-driven approach to resource management in cloud environments.

Chapter 4

Intent-Driven Resource Management

Previously, in Chapter 1, we discussed the goals of different stakeholders and how their conflicting goals make resource management a complex problem that requires careful consideration and tradeoffs. If we take a step back and look at the problem from a higher level, we make a key observation. In reality, the application owners generally only care about meeting their SLOs. The actual resource allocations required to achieve this are of secondary importance and can be abstracted away. For example, if an application owner running an online store wants to ensure that their application responds within a certain amount of time, they do not care if this is achieved by allocating one or eight cores. What matters is that the application meets its performance goals and does not cause frustrated users to leave the platform for a competitor. Other objectives such as cost play into this equation as well, but that is generally a tradeoff that can be made by adjusting the primary goals. By tolerating lower performance, the application owner can reduce costs, and vice versa.

By abstracting away the actual resources used to achieve a goal, we can instead focus on the *intent* of the application owner. Thus, by expressing their intent, the application owner can communicate their goals to the resource owner, who can then make decisions on how to best achieve those goals. This approach allows for a more flexible and adaptive resource management system that can respond to changing conditions and workloads.

The concept of intent originates in the domain of autonomous networks and software-defined networking (SDN), where it has been used to describe the desired state of a network or system. The Autonomous Networks Project at the TM Forum proposes the following definition of intent:

> "Intent is the formal specification of all expectations including requirements, goals, and constraints given to a technical system" [54].

The approach of leveraging intents for autonomously managing networks has

since been developed further in industry and is being evaluated as a core management paradigm for future telecommunication networks [55].

This definition of intent extends beyond the world of autonomous networking and can be applied to other domains, including cloud resource management.

4.1 Intent in Cloud Resource Management

While some efforts have already been made [56], [57], [58], we see a great opportunity to leverage intents in the domain of cloud resource management. By allowing application owners to express their intent instead of explicit resource requests, we can enable a more flexible and adaptive resource management system that can respond to changing conditions and workloads, even as workload composition and shared resource congestion change over time.

Such an *intent-driven* approach to resource management and workload orchestration requires a number of components to be in place. First, we need a way for application owners to express their intents in a formal and well-defined manner and communicate them to the resource owner. Second, there needs to be a way for the resource owner to interpret these intents and make decisions on how to best achieve them while maximizing their own objectives. Finally, there needs to be a feedback loop in place to ensure that the intents are being met even as conditions change over time.

These requirements coalesce into a feedback control loop that maps directly to IBM's reference architecture for autonomic computing and the traditional MAPE-K loop (Monitor, Analyze, Plan, Execute, and Knowledge) [59]. On either side of the loop's iterations, sensors allow the controller to monitor the state of the system, and effectors[2] allow the controller to make necessary adjustments. See Figure 4.1 for an illustration of the MAPE-K loop.

The modern cloud ecosystem has mature and widely adopted open-source monitoring and observability tools that can be leveraged to communicate intents and key metrics between application owners and resource owners. Prometheus [60] is a widely used monitoring toolkit and time-series database that can be used to collect and store metrics from applications and infrastructure and later relay them to the required party. Queries can be made to Prometheus using the PromQL query language, which allows for custom metrics to be defined and computed.

There are cases where the application owner wants to keep application runtime details private. This can be achieved while still supplying the resource owner with an intent by instead computing an obfuscated objective function that captures the intent of the application owner without revealing sensitive information about the application itself. Such an example could be an SLO slack ratio function. Suppose an application owner has a performance SLO K for a metric with a current observed value of k. The application owner could

[2]Another commonly used term is *actuators*.

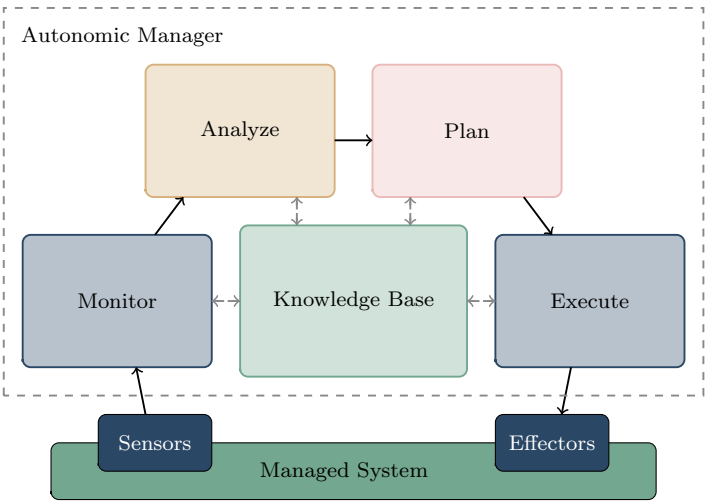


Figure 4.1: Illustration of the MAPE-K loop for autonomous system management.

then compute the slack ratio as

$$\text{Slack Ratio} = \frac{K - k}{K}. \tag{4.1}$$

As the name suggests, this function captures the slack between the current observed value and the SLO. If the workload has a 95^{th} percentile tail latency SLO of $K = 100$ ms and the current observed value is $k = 80$ ms, the slack ratio would be 0.2. That is, the application currently has 20% of its SLO target value worth of headroom. Should the observed value increase to $k = 140$ ms, the slack ratio would become -0.4, indicating that the application is now violating its SLO by 40%. This type of obfuscated objective function can be used to communicate the intent of the application owner to the resource owner without revealing sensitive information about the application itself. The only information that is revealed is whether the application is meeting its SLO or not, and by how much.

In this thesis, we build upon the idea of intent-driven resource management and explore how it can be applied to the domain of cloud resource management in **Paper III** and **Paper VI**.

4.2 The Potential of Intent-Driven Resource Management

Fully realizing the power of intents has the potential to be significant in ways not yet fully explored. As we have discussed, improving systems and distributed

systems is an optimization game where the goals and priorities of involved stakeholders are often in conflict. No solution can be optimal for all systems and stakeholders, and tradeoffs must be made.

In the past few years, the entire software development industry has been supercharged with the advent of large language models (LLMs) and generative AI. These models have shown remarkable capabilities in writing and interpreting software code, but they have yet to accelerate the domain of systems operation in the same way.

Meanwhile, as computing needs grow faster than hardware and supporting infrastructure can support them, specialized hardware accelerators are being developed to meet the needs of specific workloads [61]. This has led to a growing heterogeneity in the types of hardware available in modern data centers, which in turn has made resource management even more complex. What if intents could be leveraged to enable a new paradigm of resource management that takes advantage of the capabilities of LLMs and generative AI?

We explore this potential in **Paper VII**, where we present a vision for how intents can be leveraged to enable a new paradigm of resource management that is more adaptive, flexible, and efficient in the age of machine reasoning.

Chapter 5

Research Methods

This chapter describes the research methods used throughout this thesis. This thesis is a multi-method study that primarily uses quantitative research approaches with supporting qualitative input to form a solid foundation and to address the research objectives outlined in Chapter 1. This chapter also discusses the rationale behind the chosen methods and their relevance to the study's context. Furthermore, the papers included in this thesis are numbered in the chronological order in which they were conducted. Later studies are informed by, and build upon, the findings of earlier studies.

The investigations conducted within the scope of this thesis use research methods derived from established computer science and software engineering research practices. Specifically, the iterative nature of the *systems development research* method [62] is used as a high-level framework to guide the research process of **Paper I**, **Paper III**, **Paper IV**, **Paper V**, and **Paper VI**. The method is an iterative five-step process that starts with the *construction of a conceptual framework*. This step involves identifying the research problem and objectives, and justifying the need for a new system or solution. The second step is to *develop a system architecture* that outlines the components, modules, and their interactions. The third step is to *analyze and design the system* by creating detailed specifications, models, and prototypes. Then, the fourth step is to *build the system* by implementing the design and integrating the components. Finally, the fifth step is to *experiment, observe, and evaluate the system* by testing its functionality, performance, and usability using quantitative observations. The full iterative process is illustrated in Figure 5.1.

The experimental evaluations of the systems developed using the systems development research method were conducted using a combination of emulation and simulation techniques. In the best-case scenario, they would have been evaluated in a real-world production environment, but this was not feasible due to the lack of access and the potential risks of running experiments in a production environment. Both simulation and emulation are ways of modeling a system to study its behavior under different conditions. An emulated system is a

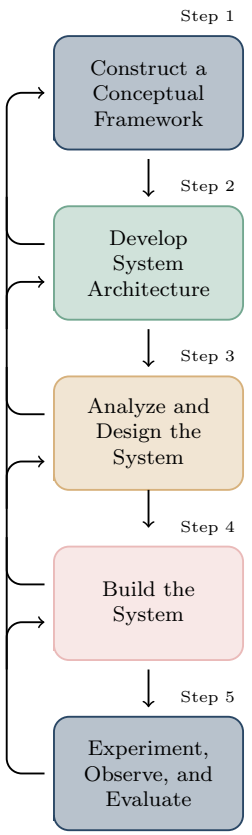


Figure 5.1: Illustration of the iterative five-step process of the systems development research method. The process starts with the construction of a conceptual framework, followed by developing a system architecture, analyzing and designing the system, building the system, and finally experimenting, observing, and evaluating the system [62].

model of a system where some components are replaced with real components, as opposed to a "pure" simulation model [63]. In most of the experiments conducted as part of the research presented in this thesis, the system under study is a real-world system, with synthetic components limited to the workloads and load generation. This closely resembles a real-world production environment, while maintaining the reproducibility and control of a system model. The emulation approach is used in **Paper I**, **Paper III**, **Paper IV**, **Paper V**, and **Paper VI**. In **Paper V**, a complementary simulation experiment evaluates the scalability of the approach by simulating a large-scale cluster.

The systematic literature survey conducted in **Paper II** explores the existing academic and industrial research landscape on hardware-level QoS enforcement features and their applications in cloud computing. It uses the *reference snowballing* method [64] to explore the literature and identify relevant papers. This is an iterative method that starts with a set of inclusion criteria and a set of initial papers that match those criteria. Then, the references and referrers of those papers are examined to determine whether they match the inclusion criteria. If they do, they are added to the set of included papers and their references and referrers are examined in the next iteration. This process continues until no new relevant papers are found (see Figure 5.2). This method was chosen over a more traditional database search approach (e.g., Kitchenham et al. [65]) for two main reasons. First, given the specific features that were being investigated, it is natural to assume there would be a few early papers describing the features that would become natural "roots" in the reference tree. Second, the reference snowballing approach has been shown to be comparable to, if not more reliable than, traditional database search methods [66].

The future outlook and vision presented in **Paper VII** were developed by taking a more qualitative approach. We had identified a general trend of increasing heterogeneity in computing infrastructure and a growing need for both horizontal and vertical insights into warehouse-scale system orchestration. To better understand the implications and the problems that arise from this trend, we needed to involve a diverse set of experts. To this end, we used a qualitative approach combining a *problem-centered expert interview* [67] process with a moderated expert discussion hosted at the 19th Cloud Control Workshop [68].

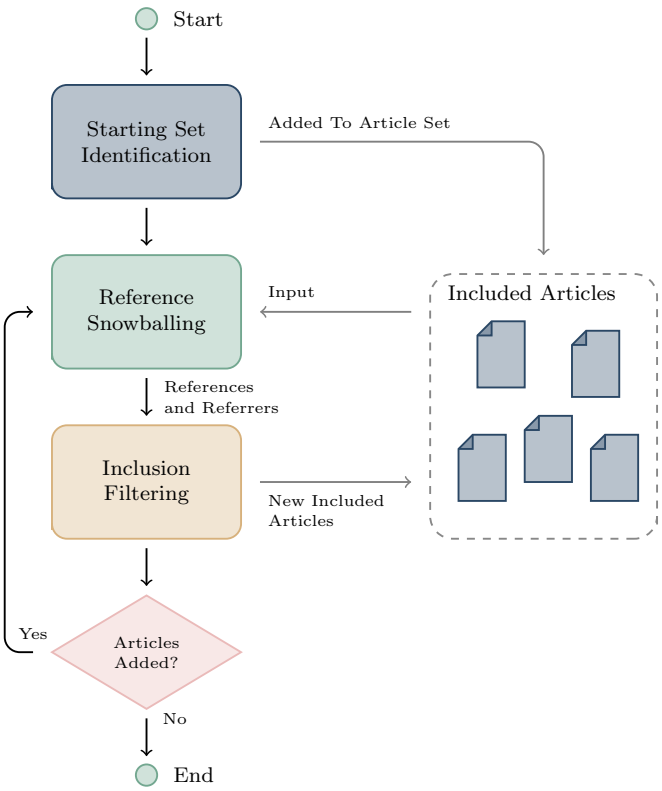


Figure 5.2: Illustration of the reference snowballing method for literature surveys. The process of investigating references and referrers is iteratively repeated as long as the set of articles that match the inclusion criteria is growing [64].

Chapter 6

Thesis Contributions

This chapter provides an overview of the primary contributions of this thesis, including a summary of the included papers and their relation to the research objectives. As we might recall from Chapter 1, the research objectives of this thesis are as follows:

RO1 Characterize hardware-level resource congestion and develop techniques for relating application-level behavior to hardware-level resource control.

RO2 Develop node-level strategies for managing shared hardware resources to improve effective utilization while maintaining workload SLOs.

RO3 Develop cluster-level resource management strategies that improve effective resource utilization by mitigating resource fragmentation while balancing the needs of application owners and resource owners.

These objectives translate into three slightly different research domains, each with its own set of contextual challenges that all need to be addressed to achieve the overall goal of improving effective resource utilization in cloud native environments. They can be summarized as follows:

- **Workload Level**—Before we can manage workloads on the node or cluster level, we need to understand how individual workloads behave and how they are affected by resource congestion, how they can be characterized, and how we can relate their behavior to the underlying hardware-level resource control (**RO1**).

- **Node Level**—Once the needs and behavior of individual workloads are understood, we can start to develop strategies for managing shared hardware resources on the node level. This includes techniques for effective node-level resource management while maintaining the SLOs of hosted workloads (**RO2**).

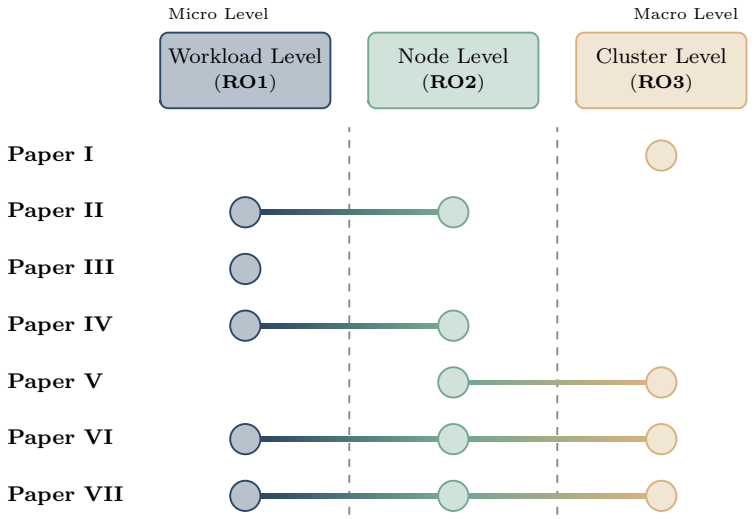


Figure 6.1: Overview of the papers included in this thesis and their relation to the research domains.

- **Cluster Level**—While effective, the local node-level resource management strategies are not enough to achieve the desired cluster-level utilization efficiency. Thus, we need to develop strategies for managing resources at the macro level, which includes techniques for workload scheduling and mitigating resource fragmentation while balancing the needs of application owners and resource owners (**RO3**).

Each individual paper focuses on one or more of these domains, as illustrated in Figure 6.1. In doing so, they each define more specific research questions which are addressed in the respective papers. These questions may be defined explicitly or, more commonly for the papers in this thesis, implicitly through the research problem being addressed.

Paper I [1] addresses **RO3** by investigating cluster-level fragmentation and ways of mitigating it using purely cluster-level mechanisms. Having found that resource fragmentation and congestion are not solely cluster-level problems, we zoom in, in **Paper II** [2], to the node and workload levels to investigate how hardware-level QoS enforcement features have been used in the literature to mitigate resource congestion and improve effective resource utilization. This primarily addresses **RO1**, while simultaneously laying a foundation for **RO2**. With the knowledge gained from **Paper II**, **Paper III** [3] proposes a novel hardware QoS enforcement strategy based on extremum seeking control [69] that focuses on the needs of a single workload, thus addressing **RO1**. The learnings of **Paper III** led to the realization that in order to make efficient hardware-level resource management decisions that respect the needs of the workloads, we need a way to reason about their resource sensitivity at runtime. Thus, **Paper IV** [38]

proposes a novel observability framework that helps identify shared resource bottlenecks at the application and node levels, which addresses **RO1** and **RO2**. With the ability to identify shared resource bottlenecks, we can start to reason about cluster-level scheduling strategies that take node-level resource pressure into account. **Paper V** [4] investigates novel cluster-level scheduling strategies that are workload aware, and how they affect the resource utilization and congestion of the cluster. In doing so, it addresses **RO2** and **RO3**. The learnings from all the previous papers are combined in **Paper VI** [70] to propose a holistic resource management strategy that deals with resource congestion and fragmentation on both the node and cluster levels, while respecting the needs of the workloads. This paper addresses all three research objectives. Finally, **Paper VII** takes a step back and looks at the bigger picture of how hardware and resource management strategies in cloud native environments are evolving, and what is needed to address the challenges of the next generation of cloud computing infrastructure. In doing so, it primarily addresses **RO3** while touching on the other two research objectives, and it also provides a vision for future research directions. Figure 6.2 illustrates the relationships between the papers and how the learnings from one paper have influenced the next.

In the following sections, a brief summary of each paper is provided, including the roles of each author and their contributions to each paper.

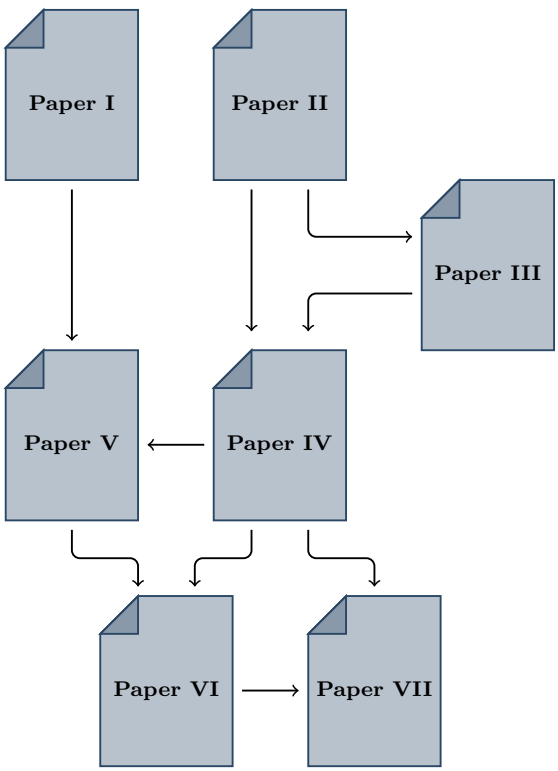


Figure 6.2: The main influential relationships between the papers included in this thesis. The arrows indicate that the learnings from one paper have influenced the next. It is important to note that implicit relationships exist by transitivity.

6.1 Paper I

Oliver Larsson, Cristian Klein, and Erik Elmroth, "The Impact of Directed Pod Eviction on Kubernetes Resource Utilization," in *Proceedings of the 2023 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, pp. 81–90, 2023.

Paper Contributions

This paper directly contributes to **RO3** by investigating the presence of cluster fragmentation and its impact on cluster utilization and resource stranding in Kubernetes environments. It shows that fragmentation is a real problem that can have a significant impact on the effective utilization of the cluster. To mitigate this problem, the paper proposes utilizing directed pod eviction as a mechanism to trigger rescheduling of workloads and simply letting the scheduler make a new placement decision for the evicted workload with the current cluster state in mind. It does this by leveraging the *Descheduler for Kubernetes* project [71] and its different eviction strategies and policies. Through emulation experiments in a Kubernetes cluster, the paper quantifies the impact of directed pod eviction on cluster utilization and resource stranding and shows that it can reduce cluster fragmentation and resource stranding. This results in the cluster being able to accommodate the same workloads with fewer nodes, thus improving the effective utilization of the cluster.

Author Contributions

The paper is an extension of Larsson's master's thesis, titled "An Evaluation of the Effectiveness of Directed Pod Eviction" [72]. The thesis was conducted under the supervision of Klein, and later extended into a conference paper within the scope of Larsson's doctoral studies with the help of Elmroth. The problem was identified by Larsson and Klein. Experiments were designed and conducted by Larsson, with data analysis and interpretation done by Larsson with feedback from Klein and Elmroth. Larsson wrote the paper with feedback from Klein and Elmroth.

6.2 Paper II

Oliver Larsson, Thijs Metsch, Cristian Klein, and Erik Elmroth, "Hardware-Level QoS Enforcement Features: Technologies, Use Cases, and Research Challenges," *ACM Computing Surveys*, vol. 58, no. 6, pp. 145:1–145:35, 2025.

Paper Contributions

This paper introduces a taxonomy of published use cases of hardware-level QoS enforcement features. The study focuses on the allocation features available through the Linux user interface for resource control [44]: cache allocation and memory bandwidth limiting. See Chapter 3 for a more detailed discussion of these features. The study contributes to **RO1** and **RO2** by expanding the understanding of how resource congestion characteristics can be related to hardware-level resource control and by providing a comprehensive overview of the state-of-the-art applications of hardware-level QoS enforcement features and how they have been used. By using the reference snowballing method [64], the study identifies 78 papers that match its inclusion criteria and analyzes them to find the use cases, their strengths and weaknesses, and the research challenges that remain to be addressed.

One of the primary findings of this study is that while hardware-level QoS enforcement features have been available for a long time, they have not been widely adopted in cloud environments. Reasons for this are identified to be, among others, a lack of insight into the workloads and their resource needs. The indirectness and complexity of tuning the hardware-level QoS enforcement features, including the interdependence of the different features, are also identified as reasons for the lack of adoption.

Author Contributions

The idea of utilizing hardware-level QoS enforcement features to improve effective resource utilization and reduce resource congestion was suggested by Metsch. In order to fully understand the state of the art and identify research gaps, Larsson suggested conducting a full literature review with the dynamics of cloud computing in focus. The methodology was identified by Larsson, while the full survey and analysis of the results were done by Larsson with feedback from Metsch, Klein, and Elmroth. The main writing of the paper was done by Larsson, with feedback from Metsch, Klein, and Elmroth.

6.3 Paper III

Oliver Larsson, Thijs Metsch, Cristian Klein, and Erik Elmroth, "ESTHER: Application-First Hardware-Level QoS-Enforcement for Cloud Native Environments," in *Proceedings of the 2025 IEEE International Conference on Cloud Computing (CLOUD)*, pp. 75–85, 2025.

Paper Contributions

This paper contributes to **RO1** by proposing ESTHER, a novel multi-target extremum seeking control-based LLC allocation and memory bandwidth limiting strategy that focuses on the needs of a single latency-critical workload. It is a feedback control strategy that observes workload SLIs and only adjusts the hardware-level QoS enforcement features to maintain a setpoint in the SLI. This demonstrates the feasibility of controlling shared resource allocations in environments that exhibit the characteristics of the modern cloud, even without explicit knowledge of the workload's resource needs.

ESTHER is evaluated experimentally in a Kubernetes environment, where it is shown to be able to maintain the SLOs of a single workload, even as the resource pressure on the node changes due to varying co-locations.

A key limitation of ESTHER is that it does not have any insights into the actual resource dynamics of any workload on the node. Rather, it relies solely on application-level SLIs to make its control decisions. We realized that in order to make more informed control decisions, we need to be able to reason about the resource sensitivity of the workloads on the node.

Author Contributions

The problem was identified by all authors together, while the idea of using extremum seeking control to control hardware-level QoS enforcement features was suggested by Larsson. System design, implementation, and experiments were done by Larsson. Analysis, interpretation of the results, and manuscript writing were done by Larsson with feedback from Metsch, Klein, and Elmroth.

6.4 Paper IV

Oliver Larsson, Thijs Metsch, Cristian Klein, and Erik Elmroth, "Workload Buoyancy: Keeping Apps Afloat by Identifying Shared Resource Bottlenecks," *Manuscript*, Umeå University, 2026.

Paper Contributions

Paper IV addresses one of the key issues identified by **Paper II** and **Paper III**, which is the lack of insight into the resource sensitivity of workloads. It proposes workload buoyancy as a novel observability framework that quantifies the resource sensitivity of workloads and identifies shared resource bottlenecks at the application and node levels. In doing so, it contributes to **RO1** and **RO2** by providing a way to reason about the resource needs of workloads that does not require any prior knowledge of the workload or its resource needs.

The paper defines resource scores, which are measures of the resource sensitivity of an individual workload to a specific resource that can be used to identify bottlenecks and make informed resource management decisions. The resource scores are combined with optional application-level SLIs to form a buoyancy metric that can be used to reason about probable resource bottlenecks and resource congestion.

Buoyancy requires no prior profiling and can be used in real time to identify resource bottlenecks and congestion. It is also extensible and can be used to reason about any resource where a resource score can be defined. As part of the evaluation of the buoyancy framework, we retested the ESTHER control mechanism proposed in **Paper III** with buoyancy as its input rather than the application-level SLI. The results showed that using buoyancy metrics as input allowed ESTHER to make better control decisions and improve stability.

Author Contributions

Larsson and Metsch identified the problem of lacking insight into the resource sensitivity of workloads. After digging deeper into the findings of **Paper II** and experimenting with the top-down microarchitecture analysis (TMA) method [73], Larsson suggested the idea of deriving a resource sensitivity metric from hardware performance counters. This led to the development of the buoyancy framework, which was designed, implemented, and evaluated by Larsson. The manuscript was written by Larsson with feedback from Metsch, Klein, and Elmroth.

6.5 Paper V

Oliver Larsson, Thijs Metsch, Cristian Klein, and Erik Elmroth, "Beyond the Limits: Flexible and Congestion-Aware Cluster Scheduling for the Cloud," in *Proceedings of the 2026 IEEE International Conference on Cloud Engineering (IC2E)*, 2026.

Paper Contributions

This paper explores cluster-level scheduling in a way that directly addresses **RO3**. It explores two different but complementary approaches to workload scheduling and rescheduling that respect the requirements of the workloads.

The first approach uses soft SLO limits that allow transient SLO violations to occur, but only if they are temporary and do not exceed a threshold. The mechanism is inspired by electrical fuses that are designed to break when the current exceeds a certain threshold. Such fuses tolerate transient overloads, but break when the overload is too long or too high. This strategy is shown to perform similarly to a traditional hard SLO limit strategy in terms of effective resource utilization and SLO violations, but with the added benefit of being able to do so with fewer rescheduling actions taken.

The second approach integrates the buoyancy framework proposed in **Paper IV** into the scheduling and rescheduling process. By using the buoyancy metrics as input, the scheduler can make better decisions that take the resource pressure of individual resources on the nodes into account and thus reduce resource congestion and improve effective resource utilization.

Author Contributions

The proposal was initially envisioned by Larsson, with the idea of using soft SLO limits being suggested by Metsch. The design was fleshed out by all authors in collaboration. The implementation, evaluation, and manuscript writing were done by Larsson, with feedback from Metsch, Klein, and Elmroth.

6.6 Paper VI

Oliver Larsson, Thijs Metsch, Cristian Klein, and Erik Elmroth, "Holistic Buoyancy-Driven Cloud Native Resource Management," *Manuscript*, Umeå University, 2026.

Paper Contributions

Paper VI combines the learnings from all previous papers to propose a holistic resource management strategy that addresses all three research objectives. It integrates the buoyancy framework from **Paper IV** into a three-component system consisting of a node-level resource management component and cluster-level scheduling and rescheduling components. A key part of the study is identifying and integrating the requirements of different stakeholders into the design of the system. The considered stakeholders are the application owners and resource owners.

The system is designed to improve resource utilization and reduce resource congestion by first making node-local allocation decisions based on the resource sensitivity of the workloads. If it is not possible to resolve the resource congestion on the node level, the system will trigger a rescheduling action that will attempt to find a better placement for the workloads on the cluster level.

The system is implemented and evaluated in a Kubernetes environment. However, it is designed to be orchestrator-agnostic and can be integrated into any cloud environment. This is also the first paper in this thesis that is evaluated at a larger scale using the Grid'5000 infrastructure, which allows for a more realistic evaluation of the system and its performance.

The results show major improvements to effective resource utilization and reduced resource congestion, while reducing resource stranding and respecting the SLOs of the workloads.

Author Contributions

This project was envisioned, implemented, and evaluated by Larsson. Metsch brought the idea of integrating stakeholders and their requirements into the design of the system, which was fleshed out together with all authors. The manuscript was written by Larsson with feedback from Metsch, Klein, and Elmroth.

6.7 Paper VII

Oliver Larsson, Thijs Metsch, and Erik Elmroth, "In the Face of Another Generational Shift in IT Infrastructure," *Manuscript*, Umeå University, 2026.

Paper Contributions

This paper takes a step back and considers the bigger picture of the challenges that arise from the increasing complexity and heterogeneity of modern cloud environments. It does so from the perspective that programmers rely more and more on generative AI to empower their programming capabilities. It argues that many of the abstractions that have been built up over the years are designed for human programmers, and that in order to take advantage of the capabilities of generative AI, we need to rethink how we design and build our systems. Furthermore, it outlines a vision for what needs to be done to address the challenges of the next generation of cloud computing infrastructure.

This paper is an opinion piece that presents a vision for the future of resource management in heterogeneous cloud computing environments. It builds upon a discussion session at the 19th Cloud Control Workshop with subsequent expert interviews being conducted to validate the findings. It contributes to **RO3** by providing a vision for future research directions and highlighting the open challenges that need to be addressed in order to improve effective resource utilization in cloud native environments in the age of generative AI.

Author Contributions

This paper was originally envisioned by Larsson with input from Metsch and Elmroth. The Cloud Control Workshop discussion session was organized by Larsson and Metsch, and the subsequent expert interviews were conducted by Larsson at the same workshop. The manuscript was written by Larsson and Metsch, with feedback from Elmroth.

Chapter 7

Related Work

This chapter provides an overview of related work in the field of resource management, focusing on three main areas: cloud resource management, hardware-level resource management, and intent-driven resource management. Each section will discuss the key contributions and findings from existing research, highlighting the gaps that this thesis aims to address by positioning itself at the intersection of these areas.

7.1 Cloud Resource Management

In the domain of resource management in the cloud, most resource management solutions approach the problem from either the perspective of the resource owner or the application owner.

7.1.1 Application Owner Perspective

Sinan [74] is a machine-learning-based resource management system that tries to identify changes in resource requirements from incoming request patterns. This allows it to proactively adjust the resource allocation to meet the changing demands of the workloads. Sage [75] also uses machine learning, but to identify the root causes of performance issues in microservice applications. It then uses this information to adjust the resource allocation of LLC and memory bandwidth to improve performance at the source of the problem. Similarly, DeepRest [76] uses deep learning to predict the resource requirements of microservices from their incoming request patterns and adjusts the resource allocation accordingly. Madu [77] manages resource allocation of microservice systems by building a model of the relationships between microservices and how they affect each other's performance. Galileo [78] uses "performance robustness certificates" that put a statistical bound on microservice tail latencies to inform resource allocation decisions. The intent-driven orchestrator [57] uses a planning component to

explore the space of possible resource allocations and find an allocation that satisfies the needs of the applications.

As opposed to the previously discussed approaches that leverage machine learning to identify the resource requirements of workloads, Ursa [79] uses an analytical model to make resource allocation decisions based on request traces.

These resource management approaches are all application-first. That is, the main focus of these approaches is to ensure that the performance of the workloads is maintained as load changes. However, they do not consider the needs of the resource owner and the overall utilization of a cluster. The exception is the intent-driven orchestrator [57], which considers the cluster utilization as a secondary objective.

7.1.2 Resource Owner Perspective

At Google, the Borg orchestrator [16], [39] has been the primary cluster management system for many years and is named as one of the predecessors of Kubernetes. Borg improves utilization and efficiency of clusters by letting application owners prioritize their workloads into different tiers. Some tiers allow for preemption of lower-priority workloads to ensure that higher-priority workloads keep running. While Borg is a holistic orchestrator, Zeus [25] is a Kubernetes-based resource management system that leverages some of the same concepts. Developed by the Chinese cloud provider NetEase, Zeus allows users to classify workloads as batch jobs to enable a rudimentary resource isolation scheme that improves cluster utilization while maintaining the performance of latency-critical workloads.

These systems are resource-owner-first and thus focus on improving the overall utilization of the cluster while ensuring that the performance of latency-critical workloads is maintained. However, they have in common that they are generally used in closed private clouds where the resource owner and application owner are part of the same organization. In public cloud environments, where the resource owner and application owner are different stakeholders with different goals, these approaches are not directly applicable.

7.2 Hardware-Level Resource Management

Due to its nature, the related work in the hardware-level resource management domain closely aligns with the taxonomy presented in **Paper II** and this discussion will be structured accordingly.

When it comes to hardware resource management, the literature can be broadly categorized into two main areas: (1) resource management approaches that treat workloads as black boxes, and thus do not consider any information about the application-level behavior of the workloads and rely solely on hardware-level metrics to make resource management decisions; and (2) application-aware resource management approaches that leverage application-level information

such as SLIs to make resource management decisions. Some application-aware approaches also consider system- and hardware-level metrics in addition to application-level information.

7.2.1 Black-Box Approaches

Ginseng [80] is one of the earliest black-box methods for LLC partitioning. It uses a market-driven approach where workloads can bid for cache resources based on their performance needs using Vickrey–Clarke–Groves (VCG) auctions. A key challenge is estimating the performance impact of cache allocation on workloads, which is addressed by using a performance model based on hardware performance counters.

Another early black-box LLC partitioning approach is dCat [81]. It classifies workloads' LLC usage behavior based on hardware metrics such as instruction retirement and instructions per cycle (IPC). Allocation changes are performed as workloads transition between different usage classes. Another LLC partitioning proposal that classifies workloads based on observed LLC characteristics is CoPart [82]. A finite state machine (FSM) is constructed from these classifications to help explore the state space and identify the optimal partitioning configuration.

The previously discussed approaches have all focused exclusively on LLC allocation. EMBA [52] conversely focuses solely on memory bandwidth limiting. Similarly, however, it classifies and clusters workloads based on their memory bandwidth usage and applies limiting schemes yielding improved performance for co-located workloads.

In the context of approaches that consider multiple resources, Satori [83] leverages Bayesian optimization to improve throughput and fairness of best-effort jobs. It partitions cores, LLC, and memory bandwidth based on the observed performance of workloads. Another learning-based multi-resource approach is DRLPart [84], which uses deep reinforcement learning to create an LLC and memory bandwidth partitioning scheme that attempts to maximize the overall IPC of co-located workloads. BALANCER [85] controls LLC and memory bandwidth by heuristically identifying operational phase changes in programs and adjusting the resource allocation accordingly using a separate controller for each resource. OLPart [36] considers the requirements of both latency-critical services and best-effort jobs when partitioning cores, LLC, and memory bandwidth with its contextual multi-armed bandit (CMAB) approach. It identifies proxy metrics available in hardware that correlate closely enough with resource requirements and uses these to make partitioning decisions.

A common theme among many of these black-box approaches is that they measure the performance of workloads in terms of IPC [36], [52], [81], [82], [84], [85], [86]. Since the IPC can vary drastically over time depending on the workload's current phase and the type of work it is doing, these approaches often become inaccurate in a dynamic and heterogeneous cloud environment where workload composition is constantly changing. Such approaches are

better suited for more static environments where workloads are well known and predictable and where pure computing throughput is the main concern, such as in high-performance computing (HPC) clusters.

So far, the black-box approaches that have been discussed reside exclusively within the context of hardware-level resource management. SpiderSense [86] is a pure hardware-level LLC partitioning approach that uses IPC and other hardware counters to classify the LLC needs of multiple latency-critical workloads, but it approaches the problem from a cloud computing perspective.

Zeus [25] differs from the other approaches in that it is primarily a Kubernetes-based cloud scheduling system that also considers hardware-level resource management. By allowing users to optionally classify workloads as batch jobs, Zeus can apply a rudimentary resource isolation scheme to improve the performance of latency-critical workloads. LLC, disk, and network are among the resources that can be isolated.

7.2.2 Application-Aware Approaches

As one of the earliest research articles to leverage hardware-level LLC partitioning, Heracles [40] monitors the performance slack of a latency-critical workload and uses that information to redistribute LLC resources to co-located best-effort workloads to improve their performance while ensuring that the latency-critical workload's performance is not negatively impacted. Similarly, Dirigent [87] improves co-located best-effort workload performance by allocating LLC resources to a latency-critical workload based on a prior profile.

Sticking with the theme of improving the performance of co-located best-effort workloads while ensuring that a single latency-critical workload meets its performance targets, PARTIES [35] has a multi-resource approach that considers cores, their frequency, LLC, and disk I/O as resource domains. It monitors the performance slack of the latency-critical workload and uses that information to redistribute resources to co-located best-effort workloads. CLITE [41] leverages Bayesian optimization to improve system utilization and best-effort job allocation while ensuring that latency-critical workloads meet their application-level performance targets by partitioning cores, LLC, and memory bandwidth, among other resources.

By classifying workloads based on their contribution to overall system performance using an offline profile, Rhythm [42] partitions LLC, among other resources, with the goal of increasing the system utilization without breaching the performance requirements of latency-critical workloads.

Another approach that improves best-effort job IPC while maintaining latency-critical service performance targets is the Markov chain isolator [88]. By modeling the performance of latency-critical workloads as a Markov chain, it can make resource allocation decisions that maximize best-effort job IPC while ensuring that latency-critical workloads meet their performance targets.

These application-aware approaches have shown that leveraging application-level information can have major benefits as it allows resources to be shifted

away from latency-critical workloads that are not currently in need of them to meet their targets. However, their reliance on offline profiling [42], [87] or restriction to a single latency-critical workload [35], [40], [87] makes them immediately unsuitable for dynamic cloud environments where workloads are constantly changing and unpredictable. Even with support for multiple latency-critical workloads, however, the challenges of cloud environments, the multiple stakeholders, their conflicting goals, and the issues of sharing information between them make these approaches difficult to apply in practice.

As was also mentioned in the context of cloud resource management, the intent-driven orchestrator [57] and Sage [75] both find allocation schemes for LLC and memory bandwidth, among other resources. However, realizing these solutions in multitenant cloud environments requires mechanisms for determining how shared-resource congestion affects the ability of individual workloads to satisfy their objectives and a way of sharing that information among stakeholders. This thesis addresses this problem through workload buoyancy and congestion-aware resource management.

7.3 Intent-Driven Resource Management

As discussed in Chapter 4, the notion of intent-driven resource management has taken hold in various domains in recent years. It was originally popularized in the context of autonomous networks and software-defined networking (SDN) and has since reached a high level of maturity in that domain [89]. Several studies have explored network traffic engineering and routing, guided by intent-driven principles, to optimize network performance and reduce congestion [90], [91], [92]. Furthermore, the intent-driven paradigm is being explored as a core management paradigm for future telecommunication networks, including 6G and beyond [93].

The networking domain is different from the computing domain and resource management systems thus operate under different requirements, constraints, and assumptions. Key ideas behind intent-driven resource management are applicable to both domains, but the nature of computing and service-based systems means their application is different in practice.

In the context of cloud computing, some intent-driven approaches have been proposed that take an application-first approach. By using reinforcement learning, Filinis et al. [58] predict load for serverless functions and autoscale resources accordingly.

Throughout this thesis, the conflicting goals of multiple stakeholders in cloud environments have been highlighted as a key challenge. Sharma et al. [94] propose an intent negotiation framework that allows for the specification of intents from multiple stakeholders and a negotiation process to resolve conflicts between them. However, the framework makes simplistic assumptions about application resource requirements and fails to consider complex and indirect resource congestion effects.

While making use of the natural language interpretation capabilities of large language models (LLMs), the CognitiveContinuum [95] framework allows users to specify intents in natural language and translates them into a priority-based resource allocation scheme. This allows non-technical users to specify intents such as "minimize cost without sacrificing reliability." The framework additionally balances tradeoffs. However, a user who is technical enough to understand which tradeoffs are necessary and possible in relation to each other is likely to want more precision in their intent specification than what this framework allows.

SLOC [56] scales services both horizontally and vertically based on user-defined SLOs in an application-first approach. It uses predictive monitoring, where a machine learning model tries to predict future SLO violations based on historical and current metrics. It then uses this information to scale the service accordingly. The intent-driven orchestrator [57] allows arbitrary intents to be specified in a declarative manner and uses a planning component to explore the space of possible resource allocations and find an allocation scheme that satisfies the needs of the applications.

7.4 Summary and Research Gap

As we have seen throughout this chapter, there is a rich body of research both in the cloud resource management domain and in the hardware-level resource management domain. In the context of cloud resource management, studies fall into two main categories: application-owner-first approaches that can be applied entirely within the context of a public cloud tenant, and resource-owner-first approaches that are typically applied in private cloud environments where the resource owner and application owner are part of the same organization. In the context of hardware-level resource management, most approaches do not consider the constraints and requirements of cloud environments, and those that do are bespoke solutions that lack deep insights into resource congestion and frameworks for sharing that information between stakeholders.

The intent-driven resource management paradigm has been explored in both the networking and computing domains, but so far lacks the ability to both consider the needs of multiple stakeholders and to reason about the complex interdependencies between application-level performance and hardware-level resource congestion in the increasingly heterogeneous and complex cloud environments.

Thus, there exists a research gap at the intersection of cloud resource management, hardware-level resource management, and intent-driven resource management (see Figure 7.1). This gap justifies research into approaches that consider the conflicting goals of multiple stakeholders in cloud environments while reasoning about the complex interdependencies between application-level performance and hardware-level resource congestion in a principled manner.

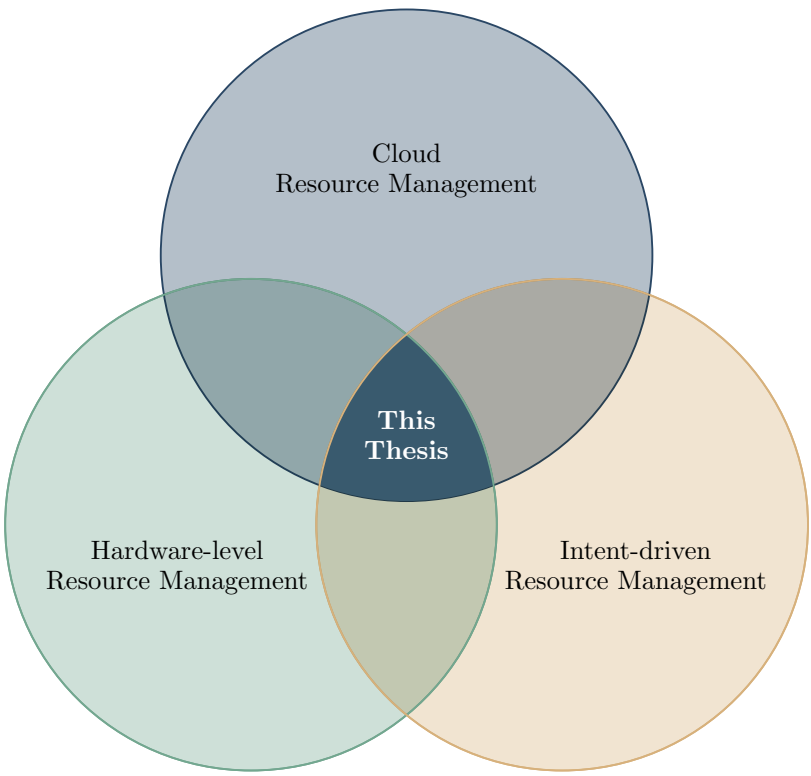


Figure 7.1: Venn diagram showing the research gap that this thesis addresses. The contributions sit at the intersection of cloud resource management, hardware-level resource management, and intent-driven resource management.

Chapter 8

Conclusions

Cloud computing environments have become increasingly heterogeneous in recent years. The complex interdependencies between different hardware and software components make it challenging to manage and optimize workloads effectively. Simultaneously, the growing demand for computing amid a limited supply of hardware and power means that we must find ways of improving the efficiency of cloud computing environments.

Congestion and noisy-neighbor effects are some of the main reasons for performance degradation in cloud computing environments [25]. Typically, these effects are mitigated by leaving a large amount of headroom in the system, leading to underutilization of available resources. By mitigating congestion and noisy-neighbor effects, we can in theory improve the efficiency of cloud computing environments, allowing for more workloads to be executed on the same hardware.

Through a series of papers, this thesis explores various aspects of shared resource congestion and workload management in cloud environments across different levels of the computing stack. As discussed in Chapter 6, this work studies and integrates the insights from congestion control and resource management in the context of individual workloads, individual worker nodes, and across entire cloud computing clusters.

By creating an understanding of the underlying causes of congestion and noisy-neighbor effects in the context of workloads (**Paper II**, **Paper III**), we develop more effective strategies for identifying and communicating the specific needs and requirements of workloads and nodes (**Paper IV**). Similarly, by studying the effects of congestion and fragmentation at the macro level of entire compute clusters (**Paper I**), we can develop more effective strategies for scheduling and allocating resources that take into account the specific needs of workloads and nodes (**Paper V**). Altogether, these contributions provide a foundation for a more holistic approach to workload management in cloud computing environments (**Paper VI**) that operates with insights across the entire computing stack, from individual workloads to entire clusters.

These findings combined provide a holistic understanding of the challenges and opportunities that exist within heterogeneous cloud computing environments. It is clear that effective workload and resource management requires insights across the entire computing stack, from individual workloads to entire clusters. Crucially, the conflicting goals of different stakeholders must be balanced to ensure that workloads are executed efficiently and reliably, while also ensuring that resource owners can maximize the utilization of their hardware. This will only become more important as the cloud computing stack becomes increasingly heterogeneous, and as the demand for computing resources continues to grow.

8.1 Future Outlook

Looking ahead, we are approaching a major shift in general-purpose cloud computing environments. As discussed in **Paper VII**, the increasingly rapid deployment of specialized hardware accelerators, such as GPUs, TPUs, and FPGAs, is quickly reshaping the landscape of cloud computing. These accelerators offer significant performance improvements for specific workloads, but they also introduce new challenges in terms of resource allocation, congestion control, and workload scheduling. The contributions of this thesis provide a foundation for addressing these challenges, and we anticipate that future research will continue to build upon these findings as these problems become more pressing.

For academics and practitioners alike, the findings of this thesis highlight the importance of managing shared resources effectively in cloud computing environments. In the academic context, we need to develop frameworks and models that enable insights from across the computing stack to be integrated into all stages of the workload lifecycle, and that account for the computing stack becoming increasingly heterogeneous. We need to be able to leverage the strengths of recent advances in machine learning and artificial intelligence to develop more effective systems without compromising their security or reliability.

For industrial practitioners, the findings of this thesis highlight the importance of balancing the goals and needs of different stakeholders. Methods should be developed that allow for more effective communication of workload and node requirements. By creating or using frameworks such as workload buoyancy that provide resource transparency and congestion insights at all levels of the computing stack, we can make better use of available resources, while also improving the performance and reliability of workloads. Ultimately, this will lead to more efficient and effective cloud computing environments that benefit resource owners and application owners alike.

Bibliography

[1] O. Larsson, C. Klein, and E. Elmroth, "The impact of directed pod eviction on kubernetes resource utilization," in *IEEE International Conference on Service-Oriented System Engineering (SOSE)*, Jul. 2023, pp. 81–90. DOI: 10.1109/SOSE58276.2023.00016

[2] O. Larsson, T. Metsch, C. Klein, and E. Elmroth, "Hardware-level QoS enforcement features: Technologies, use cases, and research challenges," *ACM Computing Surveys (CSUR)*, vol. 58, no. 6, 145:1–145:35, Dec. 2025. DOI: 10.1145/3774317

[3] O. Larsson, T. Metsch, C. Klein, and E. Elmroth, "ESTHER: Application-first hardware-level QoS-enforcement for cloud native environments," in *IEEE 18th International Conference on Cloud Computing (CLOUD)*, Jul. 2025, pp. 75–85. DOI: 10.1109/CLOUD67622.2025.00018

[4] O. Larsson, T. Metsch, C. Klein, and E. Elmroth, "Beyond the limits: Flexible and congestion-aware cluster scheduling for the cloud," in *IEEE 14th International Conference on Cloud Engineering (IC2E)*, Oct. 2026.

[5] J. Hruska and P. Maresova, "Use of social media platforms among adults in the united states—behavior on social media," *Societies*, vol. 10, no. 1, p. 27, Mar. 2020, ISSN: 2075-4698. DOI: 10.3390/soc10010027

[6] P. de Almeida, P. Fazendeiro, and P. R. M. Inácio, "Societal risks of the end of physical cash," *Futures*, vol. 104, pp. 47–60, Dec. 2018, ISSN: 0016-3287. DOI: 10.1016/j.futures.2018.07.004

[7] S. Aral, E. Brynjolfsson, and M. Van Alstyne, "Information, technology, and information worker productivity," *Information Systems Research*, vol. 23, no. 3, pp. 849–867, Sep. 2012, ISSN: 1047-7047. DOI: 10.1287/isre.1110.0408

[8] G. Mohyuddin, M. A. Khan, A. Haseeb, S. Mahpara, M. Waseem, and A. M. Saleh, "Evaluation of machine learning approaches for precision farming in smart agriculture system: A comprehensive review," *IEEE Access*, vol. 12, pp. 60 155–60 184, 2024, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2024.3390581

[9] I. Chengalur-Smith and P. Duchessi, "The initiation and adoption of client–server technology in organizations," *Information & Management*, vol. 35, no. 2, pp. 77–88, Feb. 1999. DOI: 10.1016/S0378-7206(98)00077-9

[10] A. D. Chandler, "Organizational capabilities and the economic history of the industrial enterprise," *Journal of Economic Perspectives*, vol. 6, no. 3, pp. 79–100, Sep. 1992. DOI: 10.1257/jep.6.3.79

[11] K. Appleby et al., "Oceano-SLA based management of a computing utility," in *IEEE/IFIP International Symposium on Integrated Network Management Proceedings. Integrated Network Management VII. Integrated Management Strategies for the New Millennium (Cat. No.01EX470)*, May 2001, pp. 855–868. DOI: 10.1109/INM.2001.918085

[12] Amazon Web Services, *Our origins*. Accessed: Sep. 10, 2026. [Online]. Available: https://aws.amazon.com/about-aws/our-origins/

[13] The OpenStack Authors, *Open source cloud computing infrastructure - OpenStack*. Accessed: Sep. 10, 2026. [Online]. Available: https://www.openstack.org/

[14] The Kubernetes Authors, *Production-grade container orchestration*, Kubernetes. Accessed: Sep. 10, 2026. [Online]. Available: https://kubernetes.io/

[15] B. Killen, C. Short, F. Muñoz, K. Fields, T. Bannister, and Every Contributor Across the Globe, *10 years of kubernetes*, Jun. 6, 2024. Accessed: Sep. 10, 2026. [Online]. Available: https://kubernetes.io/blog/2024/06/06/10-years-of-kubernetes/

[16] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at google with borg," in *Proceedings of the Tenth European Conference on Computer Systems*, ACM, Apr. 17, 2015, pp. 1–17, ISBN: 978-1-4503-3238-5. DOI: 10.1145/2741948.2741964

[17] Docker Inc., *Docker: Accelerated container application development*. Accessed: Sep. 10, 2026. [Online]. Available: https://www.docker.com/

[18] A. Lawson and J. Sica, *The CNCF annual cloud native survey: The infrastructure of AI's future*, Jan. 20, 2026. Accessed: Sep. 10, 2026. [Online]. Available: https://www.cncf.io/reports/the-cncf-annual-cloud-native-survey/

[19] K. Hightower and M. Balch, *Understanding configuration as data in Kubernetes*, Google Cloud Blog, Nov. 20, 2020. Accessed: Sep. 10, 2026. [Online]. Available: https://cloud.google.com/blog/products/containers-kubernetes/understanding-configuration-as-data-in-kubernetes

[20] Kubernetes SIGs, *kubernetes-sigs/cluster-api*. Accessed: Sep. 10, 2026. [Online]. Available: https://github.com/kubernetes-sigs/cluster-api

[21] F. Etro, "The economics of cloud computing," *The IUP Journal of Managerial Economics*, vol. 9, no. 2, pp. 7–22, May 2011.

[22] X. Qin et al., "How different are the cloud workloads? characterizing large-scale private and public cloud workloads," in *53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, Jun. 2023, pp. 522–530. DOI: 10.1109/DSN58367.2023.00055

[23] J. Tan and K. Thelen, "Cloud capitalism and the AI transition," *Politics & Society*, vol. 54, no. 2, pp. 184–214, Jun. 2026. DOI: 10.1177/00323292251396395

[24] J. Guo et al., "Who limits the resource efficiency of my datacenter: An analysis of alibaba datacenter traces," in *Proceedings of the International Symposium on Quality of Service*, ACM, Jun. 2019, pp. 1–10. DOI: 10.1145/3326285.3329074

[25] X. Zhang, L. Li, Y. Wang, E. Chen, and L. Shou, "Zeus: Improving resource efficiency via workload colocation for massive kubernetes clusters," *IEEE Access*, vol. 9, pp. 105 192–105 204, Jul. 2021. DOI: 10.1109/ACCESS.2021.3100082

[26] B. Reidys et al., "Coach: Exploiting temporal patterns for all-resource oversubscription in cloud platforms," in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, ACM, Mar. 2025, pp. 164–181. DOI: 10.1145/3669940.3707226

[27] N. Bashir, N. Deng, K. Rzadca, D. Irwin, S. Kodak, and R. Jnagal, "Take it to the limit: Peak prediction-driven resource overcommitment in datacenters," in *Proceedings of the Sixteenth European Conference on Computer Systems*, ACM, Apr. 2021, pp. 556–573. DOI: 10.1145/3447786.3456259

[28] X. Pu, L. Liu, Y. Mei, S. Sivathanu, Y. Koh, and C. Pu, "Understanding performance interference of i/o workload in virtualized cloud environments," in *IEEE 3rd International Conference on Cloud Computing*, Jul. 2010, pp. 51–58.

[29] T. Lorido-Botran, S. Huerta, L. Tomás, J. Tordsson, and B. Sanz, "An unsupervised approach to online noisy-neighbor detection in cloud data centers," *Expert Systems with Applications*, vol. 89, pp. 188–204, Dec. 2017. DOI: 10.1016/j.eswa.2017.07.038

[30] S.-Y. Hsieh, C.-S. Liu, R. Buyya, and A. Y. Zomaya, "Utilization-prediction-aware virtual machine consolidation approach for energy-efficient cloud data centers," *Journal of Parallel and Distributed Computing*, vol. 139, pp. 99–109, May 2020. DOI: 10.1016/j.jpdc.2019.12.014

[31] P. Sohal, M. Bechtel, R. Mancuso, H. Yun, and O. Krieger, "A closer look at intel resource director technology (RDT)," in *Proceedings of the 30th International Conference on Real-Time Networks and Systems*, ACM, Jun. 2022, pp. 127–139. DOI: 10.1145/3534879.3534882

[32] S. Gallino, N. Karacaoglu, and A. Moreno, "Need for speed: The impact of in-process delays on customer behavior in online retail," *Operations Research*, vol. 71, no. 3, pp. 876–894, May 2023, ISSN: 0030-364X. DOI: 10.1287/opre.2022.2262

[33] W. Khallouli and J. Huang, "Cluster resource scheduling in cloud computing: Literature review and research challenges," *The Journal of Supercomputing*, vol. 78, no. 5, pp. 6898–6943, Apr. 2022, ISSN: 1573-0484. DOI: 10.1007/s11227-021-04138-z

[34] M. C. Calzarossa, L. Massari, and D. Tessera, "Workload characterization: A survey revisited," *ACM Computing Surveys (CSUR)*, vol. 48, no. 3, 48:1–48:43, Feb. 2016. DOI: 10.1145/2856127

[35] S. Chen, C. Delimitrou, and J. F. Martínez, "PARTIES: QoS-aware resource partitioning for multiple interactive services," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ACM, Apr. 2019, pp. 107–120. DOI: 10.1145/3297858.3304005

[36] R. Chen, H. Shi, Y. Li, X. Liu, and G. Wang, "OLPart: Online learning based resource partitioning for colocating multiple latency-critical jobs on commodity computers," in *Proceedings of the Eighteenth European Conference on Computer Systems*, ACM, May 2023, pp. 347–364. DOI: 10.1145/3552326.3567490

[37] J. Chung et al., "Enforcing last-level cache partitioning through memory virtual channels," in *28th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, Sep. 2019, pp. 97–109. DOI: 10.1109/PACT.2019.00016

[38] O. Larsson, T. Metsch, C. Klein, and E. Elmroth, *Workload buoyancy: Keeping apps afloat by identifying shared resource bottlenecks*, Feb. 2026. DOI: 10.48550/arXiv.2602.22852 arXiv: 2602.22852[cs].

[39] M. Tirmazi et al., "Borg: The next generation," in *Proceedings of the Fifteenth European Conference on Computer Systems*, ACM, Apr. 2020, pp. 1–14. DOI: 10.1145/3342195.3387517

[40] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis, "Heracles: Improving resource efficiency at scale," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, ACM, Jun. 2015, pp. 450–462. DOI: 10.1145/2749469.2749475

[41] T. Patel and D. Tiwari, "CLITE: Efficient and QoS-aware co-location of multiple latency-critical jobs for warehouse scale computers," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb. 2020, pp. 193–206. DOI: 10.1109/HPCA47549.2020.00025

[42] L. Zhao et al., "Rhythm: Component-distinguishable workload deployment in datacenters," in *Proceedings of the Fifteenth European Conference on Computer Systems*, ACM, Apr. 2020, pp. 1–17. DOI: 10.1145/3342195.3387534

[43] D. He, L. Chen, J. Liang, C. Kang, and J. Bai, "NUMA-aware contention scheduling on multicore systems," in *16th International Conference on Intelligent Systems and Knowledge Engineering (ISKE)*, Nov. 2021, pp. 452–457. DOI: 10.1109/ISKE54062.2021.9755368

[44] F. Yu, T. Luck, and V. Shivappa, *User interface for resource control feature (resctrl)*, The Linux Kernel documentation, May 2026. Accessed: Aug. 21, 2026. [Online]. Available: https://docs.kernel.org/filesystems/resctrl.html

[45] Intel Corporation, *Intel® resource director technology (Intel® RDT) on 2nd generation Intel® Xeon® scalable processors reference manual*, Nr. 338822-001US Rev. 1.0, Apr. 2019.

[46] Advanced Micro Devices, Inc., *AMD64 technology platform quality of service extensions*, Nr. 56375 Rev. 1.03, Feb. 2022.

[47] S. Mittal, "A survey of techniques for cache partitioning in multicore processors," *ACM Computing Surveys*, vol. 50, no. 2, 27:1–27:39, May 2017. DOI: 10.1145/3062394

[48] G. Taylor, P. Davies, and M. Farmwald, "The TLB slice—a low-cost high-speed address translation mechanism," in *Proceedings of the 17th annual international symposium on Computer Architecture*, ACM, May 1990, pp. 355–363. DOI: 10.1145/325164.325161

[49] R. Iyer, "CQoS: A framework for enabling QoS in shared caches of CMP platforms," in *Proceedings of the 18th annual international conference on Supercomputing*, ACM, Jun. 2004, pp. 257–266. DOI: 10.1145/1006209.1006246

[50] A. Herdrich et al., "Cache QoS: From concept to reality in the intel® xeon® processor e5-2600 v3 product family," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Mar. 2016, pp. 657–668.

[51] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha, "MemGuard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms," in *IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, Apr. 2013, pp. 55–64. DOI: 10.1109/RTAS.2013.6531079

[52] Y. Xiang, C. Ye, X. Wang, Y. Luo, and Z. Wang, "EMBA: Efficient memory bandwidth allocation to improve performance on intel commodity processor," in *Proceedings of the 48th International Conference on Parallel Processing*, ACM, Aug. 2019, pp. 1–12. DOI: 10.1145/3337821.3337863

[53] M. Xu, L. Thi, X. Phan, H.-Y. Choi, and I. Lee, "vCAT: Dynamic cache management using CAT virtualization," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, Apr. 2017, pp. 211–222. DOI: 10.1109/RTAS.2017.15

[54] TM Forum, *Intent in autonomous networks*, Version 1.3.0 (IG1253), Aug. 15, 2022.

[55] J. Niemöller, E. Müller, M. Maggiari, and K. Maghsoudlou, "Evolving service management toward intent-driven autonomous networks," Ericsson Technology Review, Mar. 5, 2024. Accessed: Sep. 21, 2026. [Online]. Available: https://www.ericsson.com/en/reports-and-papers/ericsson-technology-review/articles/evolution-of-automation

[56] S. Nastic et al., "SLOC: Service level objectives for next generation cloud computing," *IEEE Internet Computing*, vol. 24, no. 3, pp. 39–50, May 2020. DOI: 10.1109/MIC.2020.2987739

[57] T. Metsch, M. Viktorsson, A. Hoban, M. Vitali, R. Iyer, and E. Elmroth, "Intent-driven orchestration: Enforcing service level objectives for cloud native deployments," *SN Computer Science*, vol. 4, no. 3, p. 268, Mar. 2023. DOI: 10.1007/s42979-023-01698-0

[58] N. Filinis et al., "Intent-driven orchestration of serverless applications in the computing continuum," *Future Generation Computer Systems*, vol. 154, pp. 72–86, May 2024. DOI: 10.1016/j.future.2023.12.032

[59] IBM Corporation, *An architectural blueprint for autonomic computing*, IBM White Paper, Jun. 2005.

[60] The Prometheus Authors, *Monitoring system & time series database*, Prometheus. Accessed: Sep. 21, 2026. [Online]. Available: https://prometheus.io/

[61] T. Eilam et al., "Reducing datacenter compute carbon footprint by harnessing the power of specialization: Principles, metrics, challenges and opportunities," *IEEE Transactions on Semiconductor Manufacturing*, vol. 37, no. 4, pp. 481–488, Nov. 2024. DOI: 10.1109/TSM.2024.3434331

[62] J. F. Nunamaker Jr., M. Chen, and T. D. Purdin, "Systems development in information systems research," *Journal of Management Information Systems*, vol. 7, no. 3, pp. 89–106, Dec. 1990. DOI: 10.1080/07421222.1990.11517898

[63] I. McGregor, "The relationship between simulation and emulation," in *Proceedings of the 2002 Winter Simulation Conference*, vol. 2, Dec. 2002, pp. 1683–1688. DOI: 10.1109/WSC.2002.1166451

[64] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, London England United Kingdom: ACM, May 2014, pp. 1–10. DOI: 10.1145/2601248.2601268

[65] B. Kitchenham, T. Dyba, and M. Jorgensen, "Evidence-based software engineering," in *Proceedings of the 26th International Conference on Software Engineering*, May 2004, pp. 273–281. DOI: 10.1109/ICSE.2004.1317449

[66] D. Badampudi, C. Wohlin, and K. Petersen, "Experiences from using snowballing and database searches in systematic literature studies," in *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering*, ACM, Apr. 2015, pp. 1–10. DOI: 10.1145/2745802.2745818

[67] S. Döringer, "'The problem-centred expert interview'. Combining qualitative interviewing approaches for investigating implicit expert knowledge," *International Journal of Social Research Methodology*, vol. 24, no. 3, pp. 265–278, May 2021. DOI: 10.1080/13645579.2020.1766777

[68] Cloud Control Workshop Organizers, *19th cloud control workshop*, Jun. 2026. Accessed: Jun. 26, 2026. [Online]. Available: https://cloudresearch.org/workshops/19th/

[69] A. Scheinker, "100 years of extremum seeking: A survey," *Automatica*, vol. 161, p. 111 481, Mar. 2024. DOI: 10.1016/j.automatica.2023.111481

[70] O. Larsson, T. Metsch, C. Klein, and E. Elmroth, *Holistic buoyancy-driven cloud native resource management*, Oct. 2026. DOI: 10.48550/arXiv.2610.26160 arXiv: 2610.26160[cs.DC].

[71] Kubernetes Special Interest Groups, *kubernetes-sigs/descheduler*, version v0.36.0, May 20, 2026. Accessed: Sep. 23, 2026. [Online]. Available: https://github.com/kubernetes-sigs/descheduler

[72] O. Larsson, "An evaluation of the effectiveness of directed pod eviction," Master's Thesis, Umeå University, Jan. 2023, 37 pp.

[73] A. Yasin, "A top-down method for performance analysis and counters architecture," in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Mar. 2014, pp. 35–44. DOI: 10.1109/ISPASS.2014.6844459

[74] Y. Zhang, W. Hua, Z. Zhou, G. E. Suh, and C. Delimitrou, "Sinan: ML-based and QoS-aware resource management for cloud microservices," in *Proceedings of the 26th International Conference on Architectural Support for Programming Languages and Operating Systems*, ACM, Apr. 2021, pp. 167–181. DOI: 10.1145/3445814.3446693

[75] Y. Gan, M. Liang, S. Dev, D. Lo, and C. Delimitrou, "Sage: Practical and scalable ML-driven performance debugging in microservices," in *Proceedings of the 26th International Conference on Architectural Support for Programming Languages and Operating Systems*, ACM, Apr. 2021, pp. 135–151. DOI: 10.1145/3445814.3446700

[76] K.-H. Chow, U. Deshpande, S. Seshadri, and L. Liu, "DeepRest: Deep resource estimation for interactive microservices," in *Proceedings of the Seventeenth European Conference on Computer Systems*, ACM, Mar. 2022, pp. 181–198. DOI: 10.1145/3492321.3519564

[77] S. Luo et al., "The power of prediction: Microservice auto scaling via workload learning," in *Proceedings of the 13th Symposium on Cloud Computing*, ACM, Nov. 2022, pp. 355–369. DOI: 10.1145/3542929.3563477

[78] D. Saxena et al., "Towards performance robustness for microservices," presented at the 23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26), May 2026, pp. 93–112.

[79] Y. Zhang, Z. Zhou, S. Elnikety, and C. Delimitrou, "Ursa: Lightweight resource management for cloud-native microservices," in *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, Mar. 2024, pp. 954–969. DOI: 10.1109/HPCA57654.2024.00077

[80] L. Funaro, O. A. Ben-Yehuda, and A. Schuster, "Ginseng: Market-driven LLC allocation," presented at the USENIX Annual Technical Conference (USENIX ATC 16), 2016, pp. 295–308.

[81] C. Xu, K. Rajamani, A. Ferreira, W. Felter, J. Rubio, and Y. Li, "dCat: Dynamic cache management for efficient, performance-sensitive infrastructure-as-a-service," in *Proceedings of the Thirteenth EuroSys Conference*, ACM, Apr. 2018, pp. 1–13. DOI: 10.1145/3190508.3190555

[82] J. Park, S. Park, and W. Baek, "CoPart: Coordinated partitioning of last-level cache and memory bandwidth for fairness-aware workload consolidation on commodity servers," in *Proceedings of the Fourteenth EuroSys Conference*, ACM, Mar. 2019, pp. 1–16. DOI: 10.1145/3302424.3303963

[83] R. B. Roy, T. Patel, and D. Tiwari, "SATORI: Efficient and fair resource partitioning by sacrificing short-term benefits for long-term gains," in *ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, Jun. 2021, pp. 292–305. DOI: 10.1109/ISCA52012.2021.00031

[84] R. Chen, J. Wu, H. Shi, Y. Li, X. Liu, and G. Wang, "DRLPart: A deep reinforcement learning framework for optimally efficient and robust resource partitioning on commodity servers," in *Proceedings of the 30th International Symposium on High-Performance Parallel and Distributed Computing*, ACM, Jun. 2021, pp. 175–188. DOI: 10.1145/3431379.3460648

[85] "BALANCER: Bandwidth allocation and cache partitioning for multicore processors," *Journal of Supercomputing*, vol. 79, no. 9, pp. 10 252–10 276, Feb. 2023. DOI: 10.1007/s11227-023-05070-0

[86] Z. Wei et al., "SpiderSense: Lightweight last-level cache management via time period tagging for LLC-critical workloads," *ACM Transactions on Architecture and Code Optimization*, vol. 23, no. 2, 61:1–61:24, Jun. 2026. DOI: 10.1145/3778358

[87] H. Zhu and M. Erez, "Dirigent: Enforcing QoS for latency-critical tasks on shared multicore systems," in *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, ACM, Mar. 2016, pp. 33–47. DOI: 10.1145/2872362.2872394

[88] S. Choi and H. Eom, "Markov chain-based resource-efficient and QoS-aware scheduling for latency-critical and best-effort tasks," *IEEE Access*, vol. 13, pp. 34 649–34 666, 2025. DOI: 10.1109/ACCESS.2025.3543964

[89] Y. Sharma, D. Bhamare, N. Sastry, B. Javadi, and R. Buyya, "SLA management in intent-driven service management systems: A taxonomy and future directions," *ACM Computing Surveys*, 292:1–292:38, Jun. 2023. DOI: 10.1145/3589339

[90] K. Qu, W. Zhuang, Q. Ye, X. Shen, X. Li, and J. Rao, "Traffic engineering for service-oriented 5g networks with SDN-NFV integration," *IEEE Network*, vol. 34, no. 4, pp. 234–241, Jul. 2020. DOI: 10.1109/MNET.001.1900508

[91] T. Li, C. Yang, and L. Yang, "Intent-driven QoS-aware routing management for flying ad hoc networks," in *International Wireless Communications and Mobile Computing (IWCMC)*, May 2022, pp. 1172–1177. DOI: 10.1109/IWCMC55113.2022.9824766

[92] M. A. Habib et al., "LLM-based intent processing and network optimization using attention-based hierarchical reinforcement learning," in *IEEE Wireless Communications and Networking Conference (WCNC)*, Mar. 2025, pp. 1–6. DOI: 10.1109/WCNC61545.2025.10978505

[93] Y. Wang, C. Yang, T. Li, Y. Ouyang, X. Mi, and Y. Song, "A survey on intent-driven end-to-end 6g mobile communication system," *IEEE Communications Surveys & Tutorials*, vol. 28, pp. 882–915, 2025. DOI: 10.1109/COMST.2025.3575041

[94] Y. Sharma, D. Bhamare, A. Kassler, and J. Taheri, "Intent negotiation framework for intent-driven service management," *IEEE Communications Magazine*, vol. 61, no. 6, pp. 73–79, Jun. 2023. DOI: 10.1109/MCOM.001.2200504

[95] N. Akbari, A. N. Toosi, J. Grundy, and M. A. Cheema, "CognitiveContinuum: Intent-aware orchestration using LLM and fuzzy logic," in *IEEE International Conference on Web Services (ICWS)*, Jul. 2026, pp. 279–285. DOI: 10.1109/ICWS72778.2026.00045