Benjamin Byholm

Optimizing Stateful Serverless Computing

Åbo Akademi
University

Benjamin Byholm
Born 1987

Previous studies and degrees

B.Sc., Computer Engineering, Åbo Akademi University, 2011
M.Sc., Computer Engineering, Åbo Akademi University, 2013

Optimizing Stateful Serverless Computing

Benjamin Byholm

Åbo Akademi University
Faculty of Science and Engineering
Agora, Vattenborgsvägen 3
20500 Åbo, Finland

2021

Supervisor

Prof. Ivan Porres
Department of Information Technologies
Åbo Akademi University
Finland

Reviewers

Prof. Tommi Mikkonen
Department of Computer Science
University of Helsinki
Finland

Dr.-Ing. habil. Josef Spillner
Service Prototyping Lab
ZHAW School of Engineering
Switzerland

Opponents

Prof. Tommi Mikkonen
Department of Computer Science
University of Helsinki
Finland

Dr.-Ing. habil. Josef Spillner
Service Prototyping Lab
ZHAW School of Engineering
Switzerland

ISBN: 978-952-12-4053-9 (printed)
ISBN: 978-952-12-4054-6 (digital)
Painosalama Oy, Turku, Finland 2021

Abstract

Stateful serverless computing is a new paradigm within cloud computing. It successfully incorporates state management with serverless computing. Serverless computing is a form of cloud computing where the servers necessary for performing computation have been abstracted away, leaving the choice of where and how to perform a computation solely in the hands of the cloud provider. This abstraction simplifies the programming model for the cloud user, who can focus on business logic instead of scaffolding. It also offers the cloud provider greater freedom in how to manage the involved data centers, allowing for greater utilization of available resources.

In this thesis, I propose an autonomous platform for stateful serverless computing, provide a reference design and study the involved problems while providing their solutions. I focus on optimizing the entire system from the perspective of a cloud provider in terms of efficiency, cost and quality. The platform is able to autonomously adjust the supply of computing resources to meet fluctuations in demand without unnecessary waste. I show how to manage state in an efficient manner, which reduces latency while retaining flexibility in moving computations among servers. I further show how to manage a data cache in a cost-efficient manner, trading computation for storage. I present a new model for assigning computations to servers, allowing for higher utilization of available computing resources, thereby reducing the operational expenses of the cloud provider. I also show how to quickly solve this model, allowing for continuous redistribution of computations among servers to help maintain high resource utilization.

Merging theory and practice, I evaluate my designs both analytically and empirically. For empirical evaluation, I employ computational experiments, primarily through discrete-event simulation. While this work remains in its infancy, I believe that the presented concepts can be further refined into a working production system through dedicated, practical work. Some important questions remain unanswered, but hopefully they will one day be settled.

Sammanfattning

Serverlösa datortjänster med tillståndsdata utgör en ny paradigm bland molnbaserade datortjänster. Detta område införlivar hanteringen av tillståndsdata och serverlösa datortjänster. Serverlösa datortjänster är ett delområde inom molnbaserade datortjänster där servrarna som behövs för beräkning har abstraherats bort, vilket låter leverantören av molnbaserade datortjänster avgöra var och hur beräkningar utförs. Detta förfarande förenklar utvecklingsmodellen för användaren av molnbaserade datortjänster, i och med att denne kan fokusera på verksamhetslogik i stället för infrastruktur. Å andra sidan erhåller leverantören av molnbaserade datortjänster större frihet i hanteringen av involverade datacentra, vilket tillåter en högre nyttjandegrad av tillgängliga resurser.

I denna avhandling lägger jag fram en självstyrande plattform för serverlösa datortjänster med tillståndsdata, tillhandahåller en referensutformning samt undersöker och löser de underliggande problemställningarna. Huvudinriktningen ligger på optimering av systemet som helhet, utgående från leverantörens synvinkel i fråga om effektivitet, kostnad och kvalitet. Plattformen förmår självmant anpassa utbudet av beräkningsresurser för att tillmötesgå variation i efterfrågan utan onödigt spill. Jag fastslår hur man effektivt hanterar tillståndsdata, vilket minskar latens samtidigt som flexibiliteten i att flytta beräkningar mellan servar kvarhålls. Vidare påvisar jag hur man förvaltar ett cacheminne på ett kostandseffektivt sätt genom att byta beräkning mot lagring. Jag presenterar en ny modell för att tilldela beräkningar till servrar, vilket främjar en högre nyttjandegrad av tillgängliga beräkningsresurser. På detta vis minskas leverantörens driftskostnader. Jag fastslår även hur man snabbt löser denna modell, vilket tillåter kontinuerlig omfördelning av beräkningar mellan servrar i syfte att behålla en hög nyttjandegrad av beräkningsresurser.

Genom att sammanfläta teori och praktik utvärderar jag mina skapelser såväl analytiskt som empiriskt. För empirisk utvärdering begagnar jag beräkningsexperiment, främst genom diskret händelsesimulering. Även om detta arbete fortfarande ligger i sin linda tror jag att de förevisade koncepten kan vidareutvecklas till ett fungerande produktionssystem genom hängivet, praktiskt arbete. Vissa viktiga frågor förblir obesvarade, men förhoppningsvis kommer de en dag att avklaras.

Acknowledgments

I wish to thank Prof. Ivan Porres for mentoring me in my scientific endeavors while giving me the freedom to do it my way. I also wish to thank Dr. Adnan Ashraf for a highly fruitful collaboration over the years. All my other colleagues and collaborators also deserve my gratitude, we have had many rewarding discussions and conducted highly engaging research. Finally, I wish to thank my friends and family for patiently giving me the time to work on my research, despite other plans and commitments. It is finally done. I will host a graduation party this year.

Benjamin Byholm
Åbo, April 2021

Original Publications

I Ashraf, A., B. Byholm, and I. Porres (2016). “Prediction-Based VM Provisioning and Admission Control for Multi-Tier Web Applications.” *Journal of Cloud Computing*, 5.(1), pp. 1–21. ISSN: 2192-113X.

II Ashraf, A., B. Byholm, and I. Porres (2018). “Distributed virtual machine consolidation: A systematic mapping study.” *Computer Science Review*, 28C, pp. 118–130. ISSN: 1574-0137.

III Byholm, B. and I. Porres (2014). “Cost-Efficient, Reliable, Utility-Based Session Management in the Cloud.” In *14th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*. Ed. by P. Balaji et al. IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. IEEE Computer Society, 102–111.

IV Byholm, B., F. Jokhio, et al. (2015). “Cost-Efficient, Utility-Based Caching of Expensive Computations in the Cloud.” In *23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. Ed. by M. Daneshtalab et al. Euromicro International Conference on Parallel, Distributed and Network-Based Computing. IEEE Computer Society Conference Publishing Services, 505–513.

V Byholm, B. and I. Porres (Oct. 2018). “Fast algorithms for fragmentable items bin packing.” *Journal of Heuristics*, 24.(5), pp. 697–723. ISSN: 1572-9397. DOI: 10.1007/s10732-018-9375-z.

VI Byholm, B. and I. Porres (2017a). “Optimized Deployment Plans for Platform as a Service Clouds.” In *Companion Proceedings of the 10th International Conference on Utility and Cloud Computing*. Ed. by G. Fox and Y. Chen. UCC ’17 Companion. ACM, 41–46.

VII Byholm, B. and I. Porres (2017b). *Dynamic Horizontal and Vertical Scaling of Multiple Cloud Services in Soft Real-Time*, tech. rep. 1182. TUCS.

VIII Ashraf, A., B. Byholm, and I. Porres (2015). “A Multi-Objective ACS Algorithm to Optimize Cost, Performance, and Reliability in the Cloud.” In *8th IEEE/ACM International Conference on Utility and Cloud Computing*. Ed. by O. Rana and M. Parashar. IEEE, 341–347.

Contents

Original Publications v

List of Figures ix

List of Tables xi

1 Introduction 1

1.1 Services and Sessions 1
1.2 Cloud Computing 2
1.3 Serverless Computing 4
1.4 Implementation 7
1.5 Design Considerations 9
1.6 Research Questions 12
1.6.1 How to build a stateful serverless computing platform? 12
1.6.2 How to optimize virtual machine consolidation? 13
1.6.3 How to optimize session management? 14
1.6.4 How to optimize caching? 14
1.6.5 Which optimization model suits serverless computing? 14
1.6.6 How to optimize scaling of stateful serverless systems? 15
1.6.7 How to optimize service deployments with many objectives? 15
1.7 Overview of Research 15

2 Contributions 17

2.1 Prediction-Based VM Provisioning and Admission Control for Multi-Tier Web Applications 17
2.2 Distributed virtual machine consolidation: A systematic mapping study 21
2.3 Cost-Efficient, Reliable, Utility-Based Session Management in the Cloud 24
2.4 Cost-Efficient, Utility-Based Caching of Expensive Computations in the Cloud 27
2.5 Fast algorithms for fragmentable items bin packing 30
2.6 Optimized Deployment Plans for Platform as a Service Clouds 36

2.7 Dynamic Horizontal and Vertical Scaling of Multiple Cloud Services in Soft Real-Time 40
2.8 A Multi-Objective ACS Algorithm to Optimize Cost, Performance, and Reliability in the Cloud 43
2.9 Discussion 46

3 Conclusion **49**
3.1 Future Work 51

Glossary **53**

Bibliography **55**

Paper I **61**

Paper II **85**

Paper III **101**

Paper IV **113**

Paper V **125**

Paper VI **155**

Paper VII **163**

Paper VIII **183**

List of Figures

1.1 Overallocation versus underallocation 3
1.2 System model 6
1.3 The serverless computing platform 8

2.1 Virtual machine consolidation 22
2.2 Fragmentable bin packing 31
2.3 Worst-case and asymptotic worst-case performance bounds . 33
2.4 *Example of service deployment.* 39
2.5 A Pareto frontier 45

3.1 The combined platform management algorithm 52

List of Tables

1.1 *Themes of papers constituting this thesis* 13

2.1 *Example of deployment plans* 38

I'd like to say that I knew this would happen, that it's all part of the plan for world domination.

Linus Torvalds

1 Introduction

Serverless computing is a new paradigm within cloud computing, popularized by industry. According to Baldini et al. (2017), interest in this new paradigm has been rising steadily within industry over recent years, while the academic community has paid it comparatively little attention. Back in 2011, when I first started working on this thesis, neither I, nor my colleagues had heard of the term. Baldini et al. (2017) identified Amazon as the popularizer of serverless computing through AWS Lambda in 2014, with other vendors following suit in 2016. That is not to say that the concept of serverless computing is completely new. Baldini et al. (2017) see it as emergent technology, following the increased adoption of virtualization and container techonologies.

1.1 Services and Sessions

A service, e.g. a login service, a file upload service or a video transcoding service, offers a computational resource at some cost of resources, e.g. central processing unit (CPU), random-access memory (RAM) or storage. Stateful services, such as session-based services, need reliable access to state stored between invocations. Many services that support authenticated access for multiple users are stateful by their nature, this includes modern web applications.

Session state is a form of soft state, i.e. state that can theoretically be recreated, which expires after a given duration of inactivity. Sessions exhibit workloads that are not pure online transaction processing (OLTP). That session state can be recreated does not mean that it is good to lose it. From the application's point of view, it still functions without this state, but from the perspective of the user whose session that was, the application failed, since the state was lost and it might be practically impossible to recreate an exact replica of the lost state.

This thesis focuses on the ubiquitous session-based services in its analysis and experimentation, since they constitute a real-world use case. The thesis presents strategies for dealing with session-based services in serverless computing.

1.2 Cloud Computing

To understand serverless computing, one must first understand cloud computing. A common definition of cloud computing given by Mell and Grance (2011) states that: "Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g. networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction." The economy of scale reduces cost, while the shared pool of computing resources that can be rapidly provisioned and released reduces waste. Fox et al. (2009) state that cloud computing allows converting capital expenses to operating expenses. Users no longer have to build a dedicated data center for their computing needs, nor must they rent a given number of servers from a hosting provider. Cloud computing enables utility computing, where, just as water or electricity, a theoretically unlimited amount of computing resources are available almost instantly and can be scaled up or down to meet fluctuations in demand.

This model of computing is a fairly new concept, having only existed for about a decade. While the underlying technologies, e.g. hardware virtualization, metering and distributed computing have existed for half a century and vast computing resources (relative to advancements in miniaturization, electronics and semiconductor technology) have been available on mainframes and supercomputers for almost 70 years, it is the combination of these technologies together with pay-as-you-go pricing that form the essence of cloud computing. According to Fox et al. (2009), large-scale elasticity of affordable computing resources is unprecedented in the history of information technology.

Provisioning and releasing of computing resources is known as scaling. Scaling can be done in two dimensions: horizontal and vertical. Horizontal scaling manipulates the number of computation nodes (servers) by adding or removing them, while vertical scaling manipulates individual computation nodes by increasing or decreasing their available resources. Horizontal scaling is theoretically unlimited, since it is always possible to allocate more servers to the pool. In practice, this is not always true, but that is a different problem. In contrast, vertical scaling is limited by the resources available to a server: a

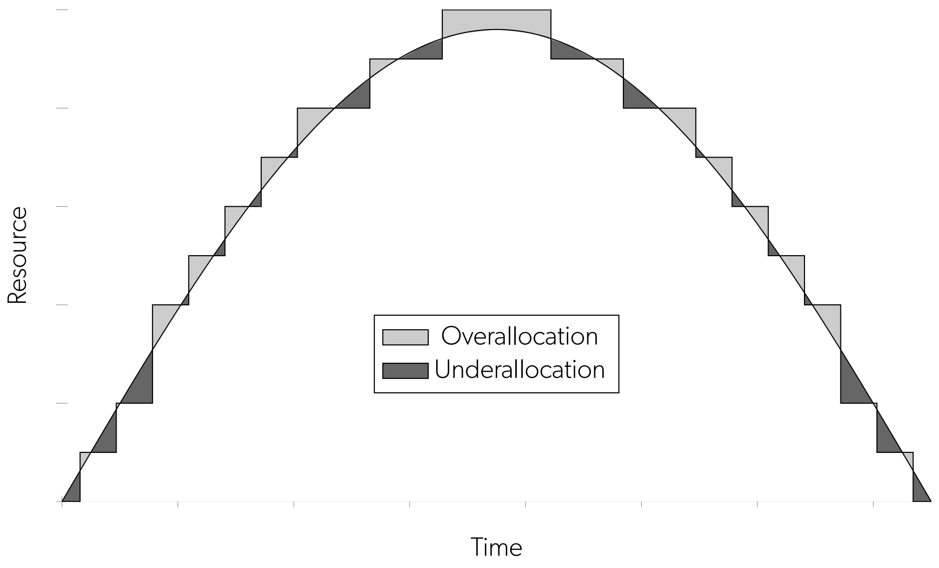


Figure 1.1: *Changes in the amount of available resources must occur in discrete steps. Overallocation is an opportunity cost, while underallocation is lost revenue.*

single server cannot offer more resources than it has. Scaling is required for supply to efficiently meet fluctuations in demand.

Figure 1.1 shows how supply can be varied to meet varying demand over time through scaling. Since computing resources are discrete, both horizontal and vertical scaling is quantized at some granularity, so scaling occurs in discrete steps. Always allocating sufficient resources to meet peak demand is unprofitable, since allocating too much resources is an opportunity cost. The same applies to allocating just enough resources to meet the minimum demand, which leads to lost revenue since the demand cannot be met. Allocating enough resources to cover mean demand could be seen as a compromise, but it is still far from optimal.

As previously stated, cloud computing offers convenient, on-demand access to shared computing resources over a network at short notice. Computing resources can be provisioned and released on the order of minutes or even seconds. Cloud computing is traditionally regarded through three paradigms: infrastructure as a service (IaaS), platform as as service (PaaS) and software as a service (SaaS). IaaS offers basic infrastructure, such as (virtual) servers in various configurations and it is the responsibility of the cloud user to manage these resources, which are billed according to a combination of duration, computational power, bandwidth and storage. On the other extreme is SaaS, which offers specific software applications to customers who may be billed on, say, a monthly basis for a software subscription. Here, the cloud user is the application owner, who may employ a lower tier, e.g. IaaS, to host the applications. The SaaS user is not aware of any infrastructure or how the software is hosted and need not manage any resources. PaaS is the odd one without a universally agreed definition, existing somewhere between IaaS and SaaS. It usually does not require explicit resource management by the PaaS user, but is more generic than SaaS in that it does not offer specific application software for end users, but provides hosting for the SaaS provider who does not desire to be an IaaS user, with all the need for explicit resource management.

1.3 Serverless Computing

What is serverless computing then? The term "serverless computing" is somewhat of a misnomer, since computation does involve servers. It is a further development of cloud computing, lying somewhere between IaaS and PaaS or SaaS, which abstracts away the servers. According to Baldini et al. (2017), the serverless paradigm is difficult to define, since there is plenty of overlap with these traditional models of cloud computing. Baldini et al. (2017) even pose an open research problem of defining the boundaries of the serverless paradigm.

As the distinctions between paradigms erode, a possible future distinction could be between server-aware and server-agnostic cloud computing. Server-aware cloud computing would be closer to IaaS, where the user has control over the servers and how computation is performed, whereas the server-agnostic model would be on the other side of the spectrum, where the cloud user leaves all operational concerns to the provider. In essence, the provider no longer directly offers infrastructure or computational resources, resigning to merely providing computation as a service, while retaining control of how to carry out any given computation. Both the server-agnostic paradigm and PaaS abstract away servers. Baldini et al. (2017) see the main difference as the ability to scale an application down to zero instances, paying for actual computation time rather than for resources.

According to Baldini et al. (2017), some forms of PaaS are serverless, but the main difference is the pricing model: PaaS charges the user for idle time, while serverless does not. Baldini et al. (2017) also regard current serverless computing systems as stateless, clearly influenced by the concept of function as a service (FaaS), but this is merely an issue of convenience. Without state, it is trivial to execute pure functions without care for where they run. Baldini et al. (2017) also pose an open research question on whether serverless computing is fundamentally stateless. As this thesis demonstrates, there is a clear answer to this question: No, serverless computing is not fundamentally stateless. Other recent works on serverless computing have also incorporated state, e.g. the video processing system by Fouladi et al. (2017) and the distributed data store by Klimovic et al. (2018).

For some applications, the server-aware paradigm with its added control is a better fit, at the cost of higher complexity with the need to actively manage the available resources, while other applications would benefit more from the server-agnostic paradigm, which alleviates the cloud user from the concerns of resource management, but offers less control. Indeed, according to Baldini et al. (2017), serverless computing poses both an opportunity and a risk to the typical IaaS user: On one hand, the programming model is simpler, since it need not deal with operational concerns. On the other hand, the user cannot directly control quality of service (QoS), monitoring, scaling and fault tolerance. It is possible to draw a parallel to resource management in programming languages: The advent of garbage collection alleviated the issue of memory management for the application programmer, offering higher productivity and ease of use, but the lack of control, nondeterminism and performance overhead might well not be suitable for the systems programmer. However, continuing this analogy, bad manual memory management may well be worse than automatic garbage collection. Returning to serverless computing, a cloud provider can take a holistic approach to the allocation of computing resources, which can improve the efficiency of a data center, since the provider can treat the entire data center as a single system and optimize it globally, whereas an individual cloud user is limited to managing the currently allotted resources. This benefits the cloud provider, who can reduce operational costs by efficient optimization and management of cloud resources.

This thesis considers stateful systems and shows how they can fit into the serverless paradigm. Its focus lies on services with hypertext transfer protocol (HTTP) sessions, but it is also applicable to multimedia services with sessions, e.g. on-demand video transcoding and streaming, as well as other kinds of sessions. Figure 1.2 illustrates the abstract architecture of such a system. At its core is the service, which performs computations in response to requests belonging to a session. Every user of a service has an associated session, maintaining some form of state between requests. Since this state must be kept somewhere, sessions have an affinity for a particular container, which implements the service to which the session belongs. The number of sessions for a given service determines how many containers implementing that service must be deployed to servers, which provide computational resources. In this context, the term container refers to any construct able to isolate a computation and limit its used resources. Virtual machines (VMs), unikernels, Docker containers, and cgroups are all examples of containers. Some containers are heavy, e.g. VMs used in basic IaaS. Others are light, such as the unikernels studied by Koller and Williams (2017). Containers could also be nested, e.g. Docker containers running in VMs.

A significant aspect of this thesis is optimization of cloud computing for

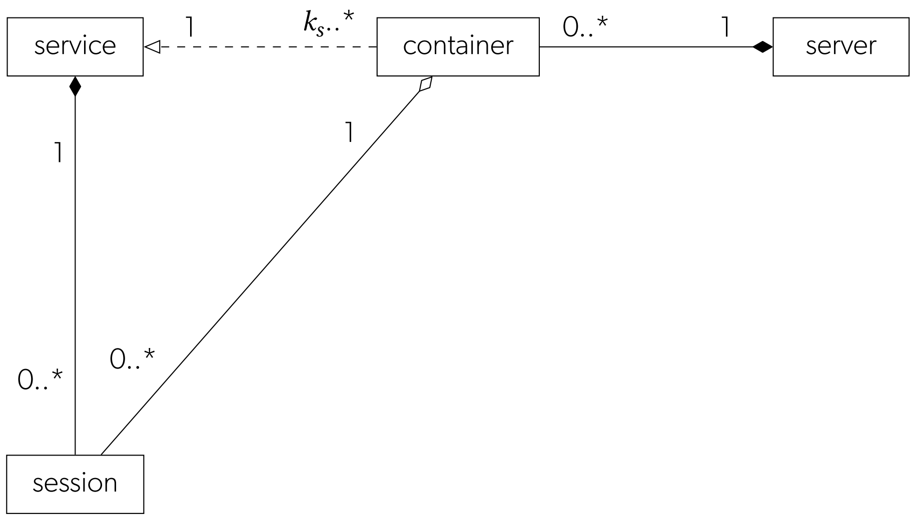


Figure 1.2: *Services are associated with sessions, determining how many containers they must deploy to servers. A session has affinity for its container. A container is an abstract construct that can isolate a computation and limit its used resources.*

providers in terms of cost, performance and reliability. Users may indirectly benefit from these gains, as part of the cost reduction can result in lower prices, and the productivity gains from not needing to manage resources may result in significant savings of time and effort. Since OLTP workloads may have high volatility in demand, which is significantly more difficult to predict compared to batch processing, decisions on how to manage the computing resources must be constantly reevaluated as the underlying premises change.

Hellerstein et al. (2019) give a critical analysis of the current state of serverless computing, analyzing the benefits and shortcomings of FaaS: Current realizations of FaaS work well for embarrassingly parallel functions, tiny independent tasks with no state, e.g. spam filters, object recognition in images and other applications of pure linear algebra. FaaS also works well for so called orchestration functions, tiny functions acting as intermediaries, offloading actual work to separate computation backends or databases. The final application of FaaS is function composition, stitching together functions from the former two categories to perform a more complex operation, similar to traditional, functional programming, but with the drawback of very high latency, lack of optimization and general cumbersomeness.

The drawbacks of FaaS listed by Hellerstein et al. (2019) are: limited lifetimes of function invocations reduces efficiency when state is employed, physical communication bottlenecks when attempting composition and the functions are spread across different physical servers, slow communication between function invocations and the lack of specialized hardware. Another look at the challenges of FaaS is given by van Eyk et al. (2018), where some of the listed challenges overlap with those mentioned by Hellerstein et al. (2019). An important, unique challenge listed is that of performance isolation: consolidating multiple FaaS functions on the same server makes it more difficult to ensure performance guarantees of individual functions, because one function may affect the performance of another. A related challenge is performance prediction, according to van Eyk et al. (2018), FaaS presents new challenges in determining the dynamic resource requirements of function invocations.

Contrary to the FaaS paradigm, I mainly deal with long-lasting stateful services involving HTTP sessions. While my system also can host and employ existing FaaS solutions, this thesis puts greater emphasis on traditional PaaS with larger, coherent applications, which incorporate state. Serverless computing is much more than FaaS and FaaS is sadly nowhere near as elegant or efficient as traditional functional programming in the vein of, say, a Haskell program. While composition of networked services may be employed, care must be taken to avoid latency dominating the computation.

1.4 Implementation

The implementation of a stateful serverless computing system warrants further examination. Figure 1.3 illustrates a reference implementation called ARVUE, developed and refined over the course of seven years. The whole system sits behind a dynamically configurable load balancer that transparently distributes all requests belonging to a session to a suitable application service, maintaining the natural session affinity for a given container and server, as illustrated in Figure 1.2, whenever possible.

When a new session arrives, the admission controller decides whether it should be admitted, rejected or deferred to the busy service according to the admission policy used in the implementation and the current occupancy of the system. When a suitable container is available, the session is automatically redirected. The busy service is infallible and never gets overloaded. If this is not acceptable, implementations are free to add scalability. A typical busy service is extremely simple, offering static content to keep a user engaged while waiting, barely requiring any resources at all and is trivially scalable.

The global controller maintains the load balancer configuration, collects statistics and manages the entire system. It forms a single point of failure, so actual implementations are advised to add redundancy as desired. The server tier is managed by the global controller, partly via the cloud provisioner, which represents the application programming interface (API) endpoint of an arbitrary IaaS cloud. This could be a public cloud, a private cloud in the provider's own data center or any variation thereof.

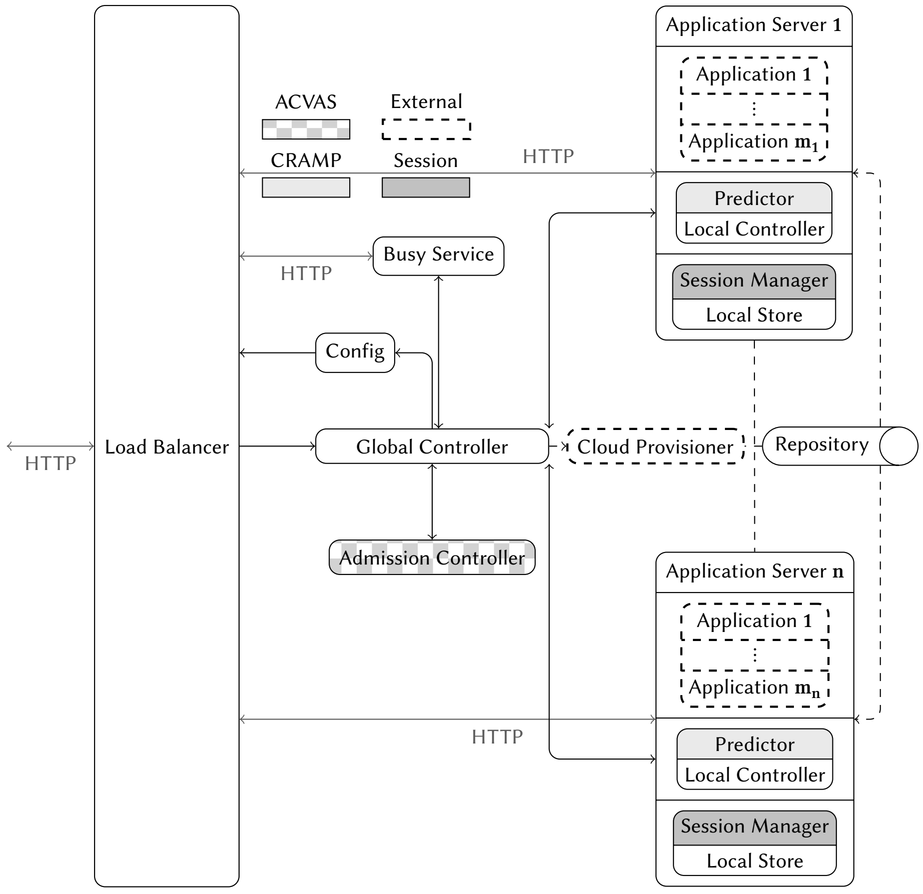


Figure 1.3: *The serverless computing platform sits behind a load balancer. At its heart lies the global controller, which makes scaling decisions for the system, configures the load balancer and directs a dynamic pool of application servers.*

Application servers run containers implementing application services. Application services are fairly large entities, which provide an API exposing several related functions. Application services are written in code by developers and compiled to a common intermediate representation, e.g. Java bytecode. A set of function callbacks for system events of interest are exposed to application services, along with a standard library for interacting with the session manager and the persistent object store in the repository.

Each application server runs a local controller, which manages everything local to the application server on behalf of the global controller. Each application server also has a local store, which is a fast but unreliable cache since it is not persistent. It could, of course, store arbitrary items, but a typical operator uses it for container images, sessions and results. Said container images and sessions are also kept in the reliable repository, which is persistent and must be infallible, so implementations must add redundancy. Each local controller has a session manager that, as the name implies, manages sessions. It is drawn separately because it is important, but would in practice be more or less integrated with the local controller. Finally, each local controller also has a predictor, which makes predictions on resource requirements local to its application server and sends these predictions to the global controller for aggregation.

1.5 Design Considerations

In any software design, there are several important considerations depending on the expected workload and use case of the system. The reference design of ARVUE is generic, but I have made it with certain use cases in mind.

In computing, workloads may be divided in two categories: batch processing and OLTP. In batch processing, data are processed in groups or batches. It is typically used for large amounts of data that require processing on a predetermined schedule, e.g. bank transactions and scientific computing. The resource demands can fairly easily be determined in advance, hence, capacity planning is straightforward. In OLTP, on the other hand, transactions are processed as they occur and users interact directly with the system. Compared to batch processing, OLTP is characterized by a significantly higher degree of volatility. A clear example of online transaction processing is a web server, where users make requests, expecting them to be processed as soon as possible. Matching the supply of computing resources to such variable demand is significantly more difficult from the perspective of a service provider. Over-allocation leads to reduced profit due to wasted capacity, while under-allocation leads to reduced profit due to the inability of serving requests in a timely manner. Baldini et al. (2017) describe the core capability of a serverless computing platform as an event processing system, so serverless computing falls within the OLTP paradigm. Hence, this thesis only considers computation in an OLTP context.

From a user's perspective, Baldini et al. (2017) argue that infrequent, but bursty workloads may be well suited for the serverless paradigm, which provides elasticity and does not charge for idle time. For a cloud provider, bursty workloads present high volatility, which is why decision algorithms for managing cloud computing resources should be fast, so that they can make informed decisions and realize them before the premises change and the decisions become outdated. Depending on the minimum billing period and provisioning delay in the underlying IaaS cloud, a small deviation from the optimum at a specific point in time is more affordable than a delay in deciding what course of action to take when managing the system resources, since deviations can be quickly corrected as long as sufficiently good decisions are made quickly enough. Most of the involved problems are very hard, if not impossible, to exactly solve in a reasonable amount of time. However, approximate solutions are easier to obtain and may well be sufficient in practice. In fact, since many practical problems involve incomplete information, predictions and estimations, there is an inherent margin of error involved in any decision. In practice, an optimal solution to an abstract decision problem may not fare significantly better than a good approximation, even if the results would be equally easy to obtain. Another way to think about this is the law of diminishing returns.

As mentioned in section 1.3, my work specifically deals with stateful services involving HTTP sessions. ARVUE has been designed with this use case in mind. However, the underlying principles are far more generic and should suit any stateful serverless computing service of OLTP nature, whether it involves HTTP sessions or not. An important problem in distributed systems is latency reduction. Paper I proposes the use of sticky sessions, which respects the natural affinity between a session and a particular container in a particular server. This reduces latency and required bandwidth, thus improving performance, compared to the traditional stateless-in-name-only approach where every stateful operation requires a remote database access. However, sticky sessions complicate load balancing, since all requests belonging to a session must be handled by the same container in the same server.

As demand grows, supply must increase to meet the demand, since under-allocation means loss of revenue. Due to the provisioning delay, horizontal scaling is not instantaneous. Neither is vertical scaling, since resizing of a container is not instantaneous. To avoid overloading the system while waiting for more resources, the admission controller is a required part of the system. The admission controller developed in Paper I can defer new sessions to the busy service, which tells the instigator to wait until a suitable container with enough spare capacity is available for the requested service. Since resources cost money, the admission controller can outright reject certain sessions if they are not deemed profitable.

Horizontal scaling should not be too aggressive: the growth factor must increase proportionally to the increase in demand to ensure the ability to meet the increased demand despite the provisioning delay, while the shrink factor should terminate unnecessary servers as soon as possible, to reduce operational cost, without producing oscillations. Paper I partly solves horizontal scaling via a control-theoretic approach based on observed resource utilization over time. It works well for meeting an increase in demand, but encounters a more difficult problem when demand subsequently decreases: Underutilized servers should be terminated to reduce operational cost, but an underutilized server with active sessions cannot be terminated without violating QoS requirements. Part of the solution to this problem is server consolidation, which has been studied in countless papers. Paper II investigates suitable options for distributed or large-scale server consolidation through a systematic mapping study. Implementations are free to choose the server consolidation strategy.

A suitable session management strategy should minimize latency, while maximizing reliability. Paper III presents the session management strategy of the system in detail. The hybrid solution of caching sessions locally while eventually adding them to the reliable store in the repository was shown to be the best for minimal latency and maximal availability at the lowest operating cost. Again, this approach also fits with the natural affinity for a given container by a given session as explained in Figure 1.2. It also facilitates termination of underutilized servers, since sessions on an underutilized server may be migrated by allowing their state to synchronize with the reliable store before terminating the server. On subsequent requests, the system will assign affected sessions to a container in another server in the same manner as if the terminated server had failed. This approach also alleviates the aforementioned problems with sticky sessions.

Various concrete systems have different bottleneck resources. A system is limited by its bottleneck resource. In relevant cases, computation is the bottleneck resource. Here, it is favorable to trade computation for storage through caching, but this cache also needs management. Paper IV looks at strategies for caching computational results in a cloud setting, which is vastly different from traditional caching of data, since the size of the cache is theoretically unlimited and there might not be a natural expiration time for the validity of obtained results, but storing every result forever is too dear.

Containers of various sizes determined by their resource needs may be deployed to servers in myriad ways. Some ways are better than other, especially when it comes to minimizing the total number of servers required, since servers cost money. Paper VI presents the best optimization model for static container placement, which is not the same as the well-known, classic bin packing, but bin packing with fragmentable items, which, in this context, is superior in every way. Paper V efficiently solves said optimization model so quickly for an ungodly number of servers and containers that the container placement can be continuously re-evaluated whenever a session enters or departs the system. Hence, Paper VII extends the optimization model to a dynamic setting, completing the goals of the whole project. The fragmentable bin packing problem studied in these papers is mainly single-objective, since its solutions can be objectively evaluated and compared in all situations, can be solved efficiently and quickly and the systems of interest have one bottleneck resource.

In some cases, the deployment problem truly is multi-objective and cannot fit in a single-objective model. Paper VIII investigates multi-objective variants of the deployment problem. Future work should focus on true multi-objective problems, but this thesis will not do that, since all solutions currently known are too slow for a dynamic setting.

A recurring theme in this thesis is the application of optimization techniques to resource management problems related to serverless computing from a cloud provider's perspective. Internet-scale applications consist of thousands of servers and services where the resource demands are highly volatile. Thus, the involved algorithms must be fast and scale to large problem instances.

1.6 Research Questions

This thesis attempts to answer several important research questions through eight published papers, where one paper answers a corresponding research question. The research questions can be categorized by partially overlapping themes, as shown in Table 1.1. A common theme across all papers is optimization: I wish to minimize the costs of operating a serverless computing system by maximizing efficiency, using as little resources as possible to meet any QoS requirements. Four of the papers included in this thesis, Papers I, VIII, VI and VII, directly deal with resource allocation. Two of them, Papers I and VII, also deal with the opposite problem of deallocation or termination. Together with Paper II, Paper VII also treats consolidation. Finally, Papers III and IV investigate caching.

1.6.1 How to build a stateful serverless computing platform?

Autonomous cloud systems are systems that require little or no external intervention to function properly. They maintain a desired configuration themselves. Such systems require observations of themselves and their environments to make informed decisions. Building an autonomous system gives rise to several important questions: Which variables can be reliably measured under various scenarios? How should they be interpreted? How should a system react to various events?

Table 1.1: *Themes of papers constituting this thesis*

	Optimization	Caching	Allocation	Consolidation	Termination
I	x		x		x
II	x			x	
III	x	x			
IV	x	x			
V	x				
VI	x		x		
VII	x		x	x	x
VIII	x		x		

Autonomous systems should make operational decisions quickly, since they need to act upon changes in the environment before the premises change too much for the decision to be meaningful: If you spend ten minutes choosing a route to catch a bus that leaves in five minutes, you just missed the bus. Not only was the excessive planning in vain, but the financial consequences of tardiness could have a negative impact exceeding that of quickly making a slightly suboptimal choice.

The many degrees of freedom offered to the service provider by serverless computing enables powerful autonomous systems, which should entail lower maintenance costs and better performance than manual operation. In Paper I, I sought to investigate the performance characteristics of an autonomous serverless computing system, how to sample observations, predict changes in demand and how to react to them. The system uses heuristics to optimize scaling decisions in an online fashion. It supports HTTP sessions through so-called sticky sessions, where all requests in a session are directed to the same application server.

1.6.2 How to optimize virtual machine consolidation?

VM consolidation aims to optimize the deployment of VMs in physical machines (PMs). As the resource demands of cloud services vary over time, at some point the system might require less computing resources than what is currently allocated. Then it is desirable to scale down the VM deployment and reduce the number of PMs. This is complicated by the need to actively migrate VMs between PMs to achieve a denser packing of VMs, consolidating operations to a lesser number of PMs. The converse problem of requiring more resources is comparatively easier to handle, since the elastic nature of cloud computing enables simply adding more servers, without a strict need for consolidating VMs.

Paper II set out to investigate what had been done in the field of virtual machine consolidation for large-scale, distributed systems. This would help identify research gaps and unexplored niches, as well as methods for solving such problems.

1.6.3 How to optimize session management?

Many cloud systems involve stateful services with sessions, where each user belongs to an active session, containing state information, such as access credentials, profile data and such. This state must be stored somewhere. Storing locally in the memory of the server currently handling the request is fast, but memory is limited and volatile. It also means that sessions cannot be easily migrated between handling servers, since that entails also moving the associated state, which can be cumbersome and complicates load balancing.

Storing globally in e.g. a shared database, does not directly suffer from this problem, however it adds increased latency and cost due to the data transfers involved. Considering the probability of data loss given different combinations of storing in several, independent storage facilities with individual failure rates, facilitates more informed decision making regarding how to manage a session with a given perceived value.

How does one construct a system that makes informed decisions about how to handle session state? What are the potential gains of such a system? What strategies for storing session state are relevant under various circumstances? I investigate this in Paper III.

1.6.4 How to optimize caching?

There are computations that require substantial effort and produce large amounts of data. Instead of redoing the computation each time its results are requested, it is possible to cache its data and reuse them for subsequent requests. However, retaining said data indefinitely quickly gets expensive. What if there are no subsequent requests for the same data? How should one develop a system that balances the cost of recomputing a data set versus storing it for a given duration so that the total cost is minimized? Paper IV addresses this.

1.6.5 Which optimization model suits serverless computing?

What representation allows modeling the allocation of sessions, services, containers and servers? One of the strengths of serverless computing is that the service provider is free to manage the allocation of computing resources, since actual servers are abstracted away from the perspective of the cloud user. In Paper VI, I explore a new optimization model that can be quickly solved by the algorithm developed in Paper V and offers many advantages compared to classic models.

1.6.6 How to optimize scaling of stateful serverless systems?

How should a serverless computing system transition between possible deployment configurations? Given a current deployment configuration and a desired deployment configuration, what way of transitioning between the current and desired deployment configuration is the best? In Paper VII, I extend the static deployment planning algorithm from Paper VI to work in a dynamic setting.

1.6.7 How to optimize service deployments with many objectives?

Many software systems must consider multiple antagonistic objectives, such as cost, performance and availability. Multi-objective optimization strives to reduce a search space to a Pareto set, consisting of a set of solutions, none of which are strictly worse than any others. The deployment of services to computation nodes is a difficult problem, which may well involve multiple antagonistic objectives. Paper VIII explores possible solutions to this problem.

1.7 Overview of Research

The publications that constitute this thesis have been selected based on their common themes and applicability to its subject. I have also authored several other papers on tangential topics, but they are not included here.

The research was conducted over several years as part of two national research projects involving industry and academia: I started with the Tekes Cloud program (2010–2013) in cooperation with Vaadin and continued with the DIMECC Need for Speed program (2014–2017) in cooperation with F-Secure.

Most of the research has been experimentally validated where applicable. As is common practice, I have mainly conducted discrete-event simulation for quantitative analysis. Some experiments have been conducted using actual prototypes, which helps in validating the simulations, but these were still under laboratory conditions. Certain aspects of the involved algorithms and problems have been formally proven, but this is not the norm.

Most software today is very much like an Egyptian pyramid with millions of bricks piled on top of each other, with no structural integrity, but just done by brute force and thousands of slaves.

Alan Kay

2 Contributions

This chapter contains a descriptive summary of the papers that form the foundation of this thesis, how they relate to the research questions in section 1.6, as well as my contributions to each paper. All papers have been coauthored with Professor Ivan Porres, who has secured funding for the research, edited drafts of manuscripts and suggested that I investigate some of the covered research areas.

2.1 Prediction-Based VM Provisioning and Admission Control for Multi-Tier Web Applications

The resource needs of web applications vary over time, depending on the number of concurrent users and the type of work performed. As the demand for an application grows, so does its demand for resources, until the demand for a bottleneck resource outgrows the supply, leading to deteriorated performance of the application. Users of an application starved for resources tend to notice this as increased latency and lower throughput for requests, or they might receive no service at all if the problem escalates. To handle multiple simultaneous users, web applications are traditionally deployed in a three-tiered architecture, where a computer cluster of fixed size represents the application server tier. This cluster provides dedicated application hosting to a fixed amount of users. There are two problems with this approach: a) if the amount of users grows beyond the predetermined limit, the application will become starved for resources. b) While the amount of users is lower than this limit, the unused resources constitute waste.

According to Vogels (2008), under-utilization of servers is a matter of concern. This inefficiency is mostly due to application isolation: a consequence of dedicated hosting. Sharing of resources between applications leads to higher total resource utilization and thereby to less waste. Thus, the level of utilization can be improved by implementing what was traditionally known as shared hosting. Dedicated VMs are fairly heavy compared to lighter container options, requiring a fairly coarse granularity, if the overhead is to remain insignificant. This can be addressed by operating at the PaaS level instead of the IaaS level, sharing PMs among VMs and VMs among lighter containers.

Capacity planning for dynamic systems is far more complicated than for static systems, since the premises change continuously and in heterogeneous, shared systems, different entities may require different amounts of various resources. Application-specific knowledge is necessary for a PaaS provider to efficiently host complex applications with highly varying resource needs. When hosting third-party dynamic content in a shared environment, that application-specific knowledge might be unavailable. It is also unfeasible for a PaaS provider to learn enough about all of the applications belonging to the users.

For example, consider a simple web service having multiple users making requests in the OLTP fashion. The number of simultaneous users, the demand, determines the resource needs of the service, or supply. Demand for a given service can fluctuate periodically over days, weeks or months: There might be fewer users during the night than during the day, less demand for summer activities during winter and vice versa. These seasonal trends are fairly easy to prepare for in advance by increasing capacity, which can then be reduced during the off-season. However, as durations shorten, the overhead of increasing and decreasing capacity by hand may outweigh the benefits. Besides cyclic trends, there might also be linear trends, such as continuous, steady growth or decline. Linear trends are also fairly easy to deal with while the durations are on the order of months or longer, but they again become challenging when the durations decrease. Sometimes there can be large spikes over short durations, e.g. when 10000 tickets for a concert are released and all are purchased within minutes. These events are the hardest to manage in a traditional data center, since they usually occur without warning, unless the traffic spike is due to a foreseeable event of which the hosting provider was notified in advance. With regard to the concert tickets, the ticket vendor might indeed notify the service provider in advance, but in the case of a startup experiencing sudden, exponential growth in the number of users or the well-known Slashdot effect, described by Kamra et al. (2004), this may well not be the case.

Having enough computational resources to always meet rare peaks implies that most of it is a waste during the other times. Conversely, preparing only for the mean amount of traffic means loss of business when demand cannot be met during the peaks. Cloud computing provides a solution to the problem of acquiring and releasing computational resources in short time, but the decisions on how and when to acquire or release resources as well as how to place VMs in PMs require automation to happen fast enough to still be relevant. Increasing scale of operations requires faster algorithms that can handle larger problems under soft real-time constraints, since the circumstances may change rapidly.

Traditional performance models based on queuing theory try to capture the behavior of purely open or closed systems. However, web applications often have workloads with sessions, exhibiting a partially-open behavior, which includes aspects of both the open and the closed model. Given a better performance model of an application, it might be possible to plan the necessary capacity, but the problem of obtaining said model remains. If the hosted applications rarely are modified it might be feasible to automatically derive the necessary performance models by benchmarking each application in isolation. This might apply to hosting first- or second-party applications. However, when hosting third-party applications under continuous development, they may well change frequently enough for this to be unfeasible.

While cloud computing allows fairly rapid provisioning of virtual machines, this alone is not sufficient to guarantee good QoS for the users of a service: As the utilization of an OLTP system approaches 1, the response time grows exponentially. A rule of thumb given by Liu (2009) states that the utilization of an OLTP system should be kept below 70 % to ensure that the response time does not exceed three times the service time of the system. Ensuring that this condition is met requires admission control in addition to load balancing. Admission control seeks to determine how many users to admit to a specific server at any given time while ensuring that said server does not become overloaded. The problem of overload is exacerbated when stateful applications are operated with session state kept locally, as with so-called sticky sessions, since the load must be balanced across entire sessions. When a server becomes overloaded, the QoS for all users of that server suffers. In an effort to please as many users as possible, the admission control solution provides a simple deferment mechanism wherein new sessions may be deferred to another service providing a simple status update until more servers have been provisioned and the deferred sessions can be admitted. This results in fewer rejected sessions and should ultimately provide a better user experience.

When this work was started in Ashraf, Byholm, Lehtinen, et al. (2012), the notion of serverless computing did not exist. The article presents a prediction-based, cost-efficient VM provisioning approach for multi-tier web applications, involving three main parts: a) a serverless computing platform called ARVUE from Ashraf, Byholm, Lehtinen, et al. (2012), b) a hybrid, reactive-proactive scaling algorithm called CRAMP from Ashraf, Byholm, and Porres (2012b) and c) a session-based adaptive admission control approach called ACVAS from Ashraf, Byholm, and Porres (2012a). The proposed approach provides automatic deployment and scaling of multiple concurrent third-party web applications on a given IaaS cloud in a serverless computing environment. It monitors and uses resource utilization metrics and does not require a performance model of the applications or the infrastructure dynamics. Its design can be seen in Figure 1.3. At its heart lies the global controller, which makes scaling decisions for the system, configures the load balancer and directs a dynamic pool of application servers that execute applications from the repository. The busy service server acts as a holding station for deferred sessions until sufficient capacity has been provisioned for them to safely be admitted to the system proper. The implementation of the CRAMP algorithm requires adding predictor units on the application servers, while addition of the ACVAS admission control approach entails the addition of a separate admission controller. External components are indicated with dashed borders. These are not part of the system proper, but serve as interaction points between the system and the outside world.

I compare the performance of three variations of the main system: a purely reactive implementation where resources are provisioned or released when certain utilization thresholds are exceeded, a hybrid, reactive-proactive variation where decisions are based on predictions of future usage weighted by the observed prediction error and finally the same system with the addition of session-based admission control. Through experiments, I find that the reactive-proactive variant performs better than the purely reactive variant, and the reactive-proactive variant with admission control outperforms both.

All variants have been developed and tested through bespoke, concrete discrete-event simulations with both synthetic and realistic workloads. The work was started before now popular cloud simulation frameworks, such as CloudSim by Calheiros et al. (2011), were published. Hence, no such frameworks were used. The first two variants, purely reactive and reactive-proactive horizontal scaling were additionally evaluated through fully functioning prototype implementations running in Amazon EC2 supplied from additional servers within the same cloud. The original articles by Ashraf, Byholm, Lehtinen, et al. (2012) and Ashraf, Byholm, and Porres (2012b) contain the results from the prototype implementations. By the time of developing the admission control approach, confidence in the quality of the simulations was high enough to cease developing prototype implementations together with their cumbersome testing infrastructure. Hence, the final variant with added admission control was only evaluated using the aforementioned discrete- event simulations. The subsequent journal article only contains simulation results.

The paper addresses RQ1 from subsection 1.6.1 by showing how to construct an autonomous, stateful serverless platform. I evaluate the entire system based on data from Dr. Ashraf's discrete-event simulations and my prototypes. The scaling algorithms for this version of the platform are centralized, as opposed to distributed, which is a drawback from a scalability perspective. However, the general principles remain applicable to distributed algorithms and the existing design with a division between global and local controllers facilitates decentralization.

The research was done in close collaboration with Adnan Ashraf, who designed most of the system and developed the algorithms through discrete-event simulations. I built the prototypes, designed experiments, analyzed data, constructed load patterns and developed the hybrid part of the reactive-proactive predictor, which produces future CPU load estimates by weighting predictions and observations according to the recent performance measurements based on the normalized root mean square error of the prediction.

2.2 Distributed virtual machine consolidation: A systematic mapping study

Cloud service providers wish to minimize the energy requirements of their data centers. According to Shehabi et al. (2016), in 2014, data centers in the United States consumed an estimated 70 TW h of electricity, corresponding to 1.8 % of the national electricity consumption. High energy consumption not only translates into high monetary cost, but also contributes to carbon emissions. According to Barroso and Hölzle (2007), production servers seldom operate near their full capacity. However, according to Fan et al. (2007), they still consume a substantial proportion of their peak power even when idle. Beloglazov and Buyya (2012) note that part of the problem is inefficient resource utilization and that energy is required not only for computation and data processing but also for cooling the equipment.

Hardware virtualization technologies enable sharing a PM among multiple VMs. This allows reliably operating several services on a single server, allowing for increased utilization, without the services and operating system having been specifically designed for this use case. Virtualization allows isolating services, thereby preventing interference and providing finer control of the resources allocated to each service. Increasing the efficiency of servers leads to energy savings by using fewer PMs.

Virtual machine consolidation refers to the act of redistributing VMs across PMs with the goal of leaving some PMs unused so that they may be shut down. For example, assuming two VMs and two PMs, each PM capable of satisfying the resource demands of both VMs, and each PM currently hosting one VM each, it is possible to move either VM to the other PM and terminate the unused

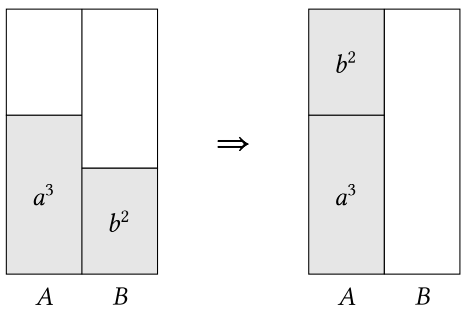


Figure 2.1: *Virtual machine consolidation refers to the act of redistributing VMs across PMs with the goal of leaving some PMs unused so they may be shut down.*

PM, thereby saving energy. Figure 2.1 illustrates this example.

This paper addresses RQ2 from subsection 1.6.2 by reviewing literature on distributed VM consolidation able to cope with large-scale distributed systems. Since VM consolidation operates on a lower level than IaaS, which is the lowest level from a cloud user's perspective, it rests firmly in the realm of data center operators, or cloud providers. Hence, this paper solely addresses cloud providers. However, this does not solely entail public cloud providers. Anybody operating a cloud, whether a fully private cloud, a hybrid solution or a public cloud is a cloud provider. The paper serves to present a comprehensive overview of the state-of-the-art in distributed VM consolidation approaches. The paper is a systematic mapping study, covering 116 unique, distributed VM consolidation approaches, narrowing them down to 19 unique approaches optimizing multiple objectives using various optimization techniques.

Most of the literature on VM consolidation has focused on centralized approaches, where observations of resource utilization are gathered from all PMs for processing in a single location where a centralized algorithm makes decisions for all PMs. This does not allow for managing multiple, geographically distributed data centers as a whole. The main drawbacks with centralized consolidation algorithms are limited scalability as well as introducing a single point of failure. In contrast, distributed consolidation algorithms use distributed algorithms or architectures, or they support multiple, geographically distributed data centers. Distributed consolidation approaches have recently become increasingly popular due to avoiding the problems of their centralized counterparts, thereby being better suited for large-scale data centers with thousands of VMs and PMs. The studied 19 distributed VM consolidation approaches form three different categories: pure, distributed VM consolidation algorithms, centralized algorithms with a distributed architecture and VM consolidation algorithms for geographically distributed data centers. A pure, distributed VM consolidation approach uses a distributed algorithm to find migration plans optimizing the placement of VMs in PMs. The second category uses a centralized algorithm, but the system itself is distributed. The third category consists of approaches that extend centralized algorithms to support VM consolidation in multiple, geographically distributed data centers.

This study found that most of the included approaches are decentralized algorithms where the systems are divided into autonomous regions, which may coordinate decisions among themselves. Out of 19 studied approaches, 14 use pure, distributed VM consolidation, 2 present centralized algorithms with a distributed architecture and 3 present VM consolidation approaches for geographically distributed data centers. Since the problems are NP-hard, and require automated decision making, the majority of approaches (80 %) employ heuristics and metaheuristics for optimizing a single objective function. The most popular algorithm type is distributed or coordinated local search, employed by 6 approaches. The vast majority of studies (14) are based on offline optimization. The nature of 3 of the studies was indeterminate. Only 2 approaches were confirmed as using an online algorithm. While offline optimization algorithms theoretically can give far better solutions than online algorithms, since they have access to more data, this is often not the case in practice, since it requires complete knowledge of the problem, all variables and possible future events. In contrast, an online optimization algorithm makes decisions based on currently available, limited information in the current state. This paradigm readily lends itself to distributed algorithms and is therefore an interesting subject of further research.

Analysis of the employed optimization algorithms revealed that 6 of the studies used single-objective optimization, 11 used multi-objective (2–3) optimization with an aggregate objective function, while 2 used many-objective (4–5) optimization with an aggregate objective function. Aggregate objective functions are a popular means of employing a single-objective optimization algorithm on a multi-objective problem. They reduce complexity and improve the execution speed by reducing the search space. However, aggregate objective functions require weights to be assigned to the objectives, which is a subjective process requiring detailed domain-specific knowledge to achieve a desired outcome. None of the studied approaches use pure, multi- or many-objective optimization. This forms a clear research gap.

Most proposed approaches (16) are evaluated empirically though simulations, with prototype implementations being rare, due to the high cost and complexity of conducting large-scale tests of actual implementations. Only 3 approaches were evaluated using prototype implementations. There are hardly any comparative analyses between different approaches, partly due to lack of readily available simulation code or other artifacts. Only one studied paper had a comparison of the results from another studied paper.

The research was done in close collaboration with Adnan Ashraf, who designed the review protocol, conducted the literature search and wrote most of the paper. I suggested the subject of the study, independently reviewed papers and aided in analyzing the findings. The study follows a review protcol developed by Dr. Ashraf, wherein all papers selected for inclusion were independently reviewed by both him and me with the goal of minimizing bias. The protocol specifies that disagreements should be resolved through meetings and discussions until consensus is reached. Thus, the results of the study are by definition a joint statement, in accordance with the fundamental principles of the protocol.

2.3 Cost-Efficient, Reliable, Utility-Based Session Management in the Cloud

According to Ling et al. (2004), session state is a form of soft state that expires after a certain time interval has passed since the last request in a given session. Session state is especially important in interactive web applications that provide a rich user experience. Services can store session information in different storage systems, e.g. local memory, a flat file system, a distributed cache, or a database. Each session store has different characteristics, e.g. available capacity, durability, and reliability.

Session state is also how an application maintains a user's workflow. If this state is lost, the user will perceive an application failure. Hence, session state must be reliably stored with high performance. Although the most efficient way of storing session state from a performance point of view is to keep it locally in memory at the application servers, this alone is not reliable and does not scale well, since it requires session affinity, where where users are assigned to a specific server for the duration of a session. Session affinity violates the principle of separation of concerns, as application servers become responsible not only for application logic, but also for storing session state. Session affinity also complicates load balancing, since different classes of requests may have different resource requirements and service times, but the load can only be balanced at the session level and not at the request level.

In the context of a commercial digital service, successful handling of a session can lead to some revenue for the service provider. However, in the context of cloud computing, where the application provider pays for computing resources per use, each session store also has a different cost per transaction, reducing the expected profit. The question that I answer in this article is: How can an application provider maximize the expected profit by minimizing session costs?

At any given moment, a session management system may decide to store a session in one or more storage subsystems, or it can decide to delete a session. These decisions will affect the reliability, revenue, and cost of the whole application. They must account for hardware constraints, such as how many sessions fit in a store, e.g. local memory, or how many sessions can be written to a slow store, e.g. a database, in a particular time interval. Given any number of sessions with different expected revenues and resource requirements, e.g. size, a session management system should determine an efficient allocation of sessions to stores, meeting the reliability requirements while maximizing profit. For example, at any given moment, the system should decide which sessions to keep in volatile, but fast, RAM on the application server, which sessions to keep in a reliable, but slower, remote store and which to drop entirely. I consider that there is a finite amount of RAM available, operations cost money and the expected revenues differ between sessions. A cost-efficient solution to this problem makes previously unprofitable services profitable and increases profit in others.

Ling et al. (2004) note that retrieval of session state should be reliable and fast, but due to its transient nature, session state does not require full atomicity, consistency, isolation and durability (ACID) semantics. Instead, many network services can trade consistency for availability by employing basically available soft state with eventual consistency (BASE) semantics, which are weaker than ACID. With soft state, a system writes the state to several independent data stores and retrieves it from one of them. Thus, data are lost only if all copies are lost. However, soft state is not free from drawbacks. For example, as noted by Goldberg (2009), the level of consistency, and thereby reliability, is determined by the frequency of the update messages. A higher frequency of update messages leads to increased cost and overhead. Increased overhead means greater delay. Any delay when retrieving session state will increase the service time for all requests, as processing of a request cannot proceed without the required session state.

In this paper, I develop a general utility model to solve this problem that takes reliability into account and works in other contexts than e-commerce. I achieve this by applying von Neumann-Morgenstern lotteries, originally developed by von Neumann and Morgenstern (1953), to represent decisions and outcomes for storing state, which facilitates applying a utility function to make informed decisions and maximize the expected utility of said decisions.

A von Neumann-Morgenstern lottery is a set of mutually exclusive outcomes with associated probabilities that sum to one. For example, $L = 0.20A + 0.80B$ denotes a lottery L where the probability of outcome A is $P(A) = 0.20$ and the probability of outcome B is $P(B) = 0.80$. A rational agent obeying the axioms of the von Neumann-Morgenstern utility theorem has a utility function u, assigning a real value $u(A)$ to every possible outcome A, so that for any two lotteries L and M, a preference of M to L is equivalent to a greater expected utility of M to L. By constructing a rational agent, I model its choices as lotteries and compare their expected utilities, facilitating the choice of the most desirable expected outcome.

Utility works as an abstraction that captures the perceived gain from obtaining something, its utility. The choice of utility function can affect the risk-seeking or risk-avoiding nature of the agent, in addition to the overall perceived utility of the outcomes of a given lottery. Representing utility in terms of money provides a back-of the envelope cost-benefit analysis and can produce an agent that tries to minimize loss and maximize profit. This allows for rational decision making in an autonomous serverless computing system.

I combine my utility model with a Markovian reliability model for multiple session stores, giving the risk of data loss for a session. Based on the model, I propose a system for utility-based session management and compare 9 different session management policies, each with 2 different revenue models: constant revenue and variable revenue per session, in 3 different businesss scenarios: low, medium and high revenue-cost ratio, through discrete- event simulations and analyze the results. The results show that utility-based session management increases cost-efficiency, but if the ratio between expected revenue and cost of sessions is too high, the difference will be negligible. However, with a low ratio of expected revenue to cost, such as an advertisement- funded or freemium web application, utility-based session management means the difference between the application provider operating at a net loss or profit.

The paper uses a knapsack model combined with von Neumann-Morgenstern lotteries to optimize the expected utility of a session-based cloud system by choosing whether, when and where to store session data under uncertainty. The knapsack problems are a family of NP-hard optimization problems that are at least as hard as the hardest problems in NP. In practice, this means that exact solutions are extremely computationally expensive. Since the intended use case needs solutions quickly, because the premises might change, I decided to use a greedy heuristic for the problem. In the experiments, this heuristic is compared to a slow, unrealistic algorithm serving to give an upper bound on how well a policy would perform under ideal conditions, facilitating a further comparison of policies without the impact of the specific greedy heuristic I chose to use.

In the knapsack problem, there is a given set of n items with associated profits p and resource requirements r. The objective is to pick a subset of items that maximizes the total profit without exceeding the capacity R of the knapsack:

$$\begin{aligned} \text{maximize} \sum_{i=1}^{n} p_i x_i \\ \text{subject to} \sum_{i=1}^{n} r_i x_i \leq R \\ x_i \in \mathbb{B} \end{aligned}$$

The binary knapsack problem can be extended to multiple resources in a variant known as the multiple-choice multi-dimension knapsack problem (MMKP). In this problem there are n groups, each having l items. The objective is to pick one item from each group, yielding the highest profit without exceeding the m resource constraints:

$$\begin{aligned}
\text{maximize} \sum_{i=1}^{n} \sum_{j=1}^{l_i} p_{ij} x_{ij} \\
\text{subject to} \sum_{i=1}^{n} \sum_{j=1}^{l_i} r_{ijk} x_{ij} \leq R_k \\
\sum_{j=1}^{l_i} x_{ij} = 1 \\
x_i \in \mathbb{B} \\
k = 1, \dots, m
\end{aligned}$$

The chosen greedy heuristic is a straightforward extension of the well-known greedy heuristic for a classic knapsack problem, where items are chosen in the order of highest value-size ratio first. In the MMKP, there can be multiple resources, which are reduced to one value using the aggregate resource consumption measurement developed by Toyoda (1975), after which items are picked in the same manner as with the classic knapsack. In practical use, in a scenario where the details of the employed packing algorithm are important, a more advanced but equally fast algorithm would be worth investigating.

I simulated a session management system with two storage levels: an unreliable, but fast session store contained in RAM on the application server and a reliable, but slow session store on a remote server. Two session stores yield four possible decisions for managing a session: store in none, store locally, store remotely and store in both. I constructed von Neumann-Morgenstern lotteries over the expected profit from each of these alternatives, which formed the multiple choices for the MMKP, which I solved using a greedy heuristic. Prof. Porres suggested that I investigate how von Neumann-Morgenstern lotteries could be applied to session management. The paper addresses RQ3 from subsection 1.6.3.

2.4 Cost-Efficient, Utility-Based Caching of Expensive Computations in the Cloud

In this paper, I present a decision model for caching expensive computations that produce large amounts of data in a cloud. This model is applicable to cloud services that produce vast quantities of data that can be reused in subsequent requests for the same data.

Caching in itself is old news, but certain aspects change in a cloud context. Traditionally, caching strategies pertain to a fixed-size cache, smaller but faster than the primary storage, with the aim of reducing data movement. For example, Pérez et al. (2018) apply caching to containers in a container-based serverless system with an application to image processing. However, when it comes to the actual results of a computation, given the theoretically limitless storage resources available in the cloud, the size of a cache need not be limited in practice, but data stored in the cloud come at a cost, which increases over time. Hence, the central question for a caching strategy in this context is whether caching the result of a given computation is worth the associated cost, as opposed to running the computation again, and for how long that result should be cached. The resulting data must always be transferred to the requester. Thus, the primary purpose of this type of cache is to reduce operational cost, instead of latency or data movement.

As I show in this paper, an efficiently managed cache can significantly reduce the operational cost of such a system. As a concrete example, I study its application to a cloud-based, on-demand video transcoding service, previously developed by my coauthors. While focusing on caching the results from video transcoding in particular, the paper answers RQ4 from subsection 1.6.4 in a generic fashion, directly making it applicable to caching many other expensive computations in a cloud setting.

An important detail is that the results of a video transcoding operation can be sent to the requester on the fly, while continuing to process the remainder of the video. This eliminates latency to the extent that it need not be directly accounted for in the underlying models, simplifying the decisions. I presume that this holds true for many other practical applications involving vast amounts of data requested in an OLTP fashion, since the data must be transferred to the requester.

A video transcoding service converts digital videos from one format to another. Formats differ in several aspects, mainly in resolution, bitrate and encoding. This discrepancy is especially problematic on mobile devices, which may have hardware support for decoding certain encodings, limited bandwidth and small displays. If a client device does not support a given format, or desires another, the video must be transcoded. For the uninitiated, Vetro et al. (2003) offer an overview of video transcoding, but for the purposes of this paper, it is sufficient to know that video transcoding generally is very expensive in terms of required computation.

By storing a transcoded video for an additional amount of time after the client device no longer wants the video, one can avoid repeating expensive transcoding operations, thereby saving relatively large amounts of money.

After this additional time, the circumstances may be reevaluated to make a new decision on whether or not to continue storing the video.

In the context of a pay-per-use cloud computing infrastructure, each transcoding operation has a monetary cost due to use of computation resources, while video storage has a cost based on the amount of data and time to be stored. To reduce the operating costs of the service, one must decide when and for how long each transcoded video should be cached in the storage. A service that stores data that will not be requested in the future will incur unnecessary storage costs. On the other hand, a service that discards data too eagerly is susceptible to incurring unnecessary computing costs.

In this paper, I study the aforementioned problem and propose a decision model for cloud-based caches with the objective to reduce operating costs. This paper applies utility theory to decision making through von Neumann-Morgenstern lotteries. My utility model for decision making requires three unknown parameters: 1. the storage duration t, 2. the mean number of arrivals $m(t)$ over the storage duration 3. and the popularity distribution p_i of cached objects o_i in the system. I present a natural way of obtaining a good value for the storage duration t, having nice properties that help evaluate the performance of the decision algorithm. I obtain the number of arrivals $m(t)$ over the storage duration t by solving a sub-problem consisting of predicting future arrival counts through singular value decomposition, as outlined by Shen and Huang (2008). Finally, I employ the Simple Good-Turing frequency estimator, developed by Gale and Sampson (1995), to estimate the relative popularity p_i of each cacheable object in the system. I evaluate the decision making approaches through discrete-event simulations and find that the proposed approach offers 72 % lower cost compared to always storing all requested objects.

Based on Jokhio et al. (2013), I determine the storage duration t as the constant τ, which is the cost of transcoding divided by the cost of storing data and the transcoding rate. Assuming that I can model requests arriving to the system as an inhomogeneous Poisson process with mean $m(t)$, I use Poisson splitting to derive the mean of requests for each video in a given format o_i as $p_i m(t)$. For any object o_i, the system can choose whether to delete or store it and each decision has two possible outcomes: either additional requests arrive, or they do not. From this I form two von Neumann-Morgenstern lotteries and conclude that one should opt to store when $p_i m(\tau) \geq 1$, i.e. whenever at least one more request for a video in a given format in time τ is expected.

To obtain the mean arrival rate of requests for the entire system $m(t)$, I construct a fast predictor using truncated singular value composition, developed by Baglama and Reichel (2005), to quickly reduce the dimensionality of the underlying data, allowing the use of simple linear regression to quickly extrapolate its individual major components. The system may then predict the future number of arrivals over various intervals.

Determining the popularity p_i of an object o_i is rather tricky: Using the obvious, empirical maximum likelihood estimator works well for popular objects, but popular objects are not interesting, since they always have a high likelihood of being requested, thus benefiting from being stored. The (currently) unpopular objects constitute the matter of concern, since they are less likely to be requested within time τ and might be worth deleting. To solve this problem, I decided to use the Simple Good-Turing frequency estimator, presented by Gale and Sampson (1995), which accounts for unobserved events. One must also keep in mind that this is a dynamic system, in which the popularity of videos changes over time as new videos are added and others fall into oblivion. To handle this, I apply a sliding window to the historical data.

I evaluate the resulting decision algorithm through discrete-event simulations involving 10000 videos with random sizes following a double Pareto-lognormal distribution, with video popularity following a truncated Pareto distribution giving a long tail. I calculate the cost of operating the simulated system over one year using four decision algorithms: my utility-based algorithm, the same algorithm with perfect information of the future — to allow comparing the accuracy of the estimation part, a naïve algorithm which always stores each video indefinitely, as well as the previous approach developed by my coauthors. I conclude that the accuracy of my predictor appears very high, since I saw an improvement of only 4 % lower cost when using perfect information, as opposed to the described estimations and predictions. My decision algorithm clearly outperforms all other tested algorithms.

This paper is an extension of a previous work on video transcoding done by my colleagues Jokhio et al. (2013), which developed the method of determining the storage duration constant τ. I applied von Neumann-Morgenstern lotteries to develop the utility model that decides whether or not to store a video. To this end I also developed the predictor that determines the expected number of future requests for any given video by combining a predictor for the number of requests to the system with a frequency estimator for rare events. I designed and conducted the experiments, analyzed the results and wrote most of the paper.

2.5 Fast algorithms for fragmentable items bin packing

Unlike the other papers that constitute this thesis, this paper is not about cloud computing. In this paper I develop fast algorithms for solving a new class of abstract packing problems known as bin packing with fragmentable items. Bin packing with fragmentable items is a variant of the classic bin packing

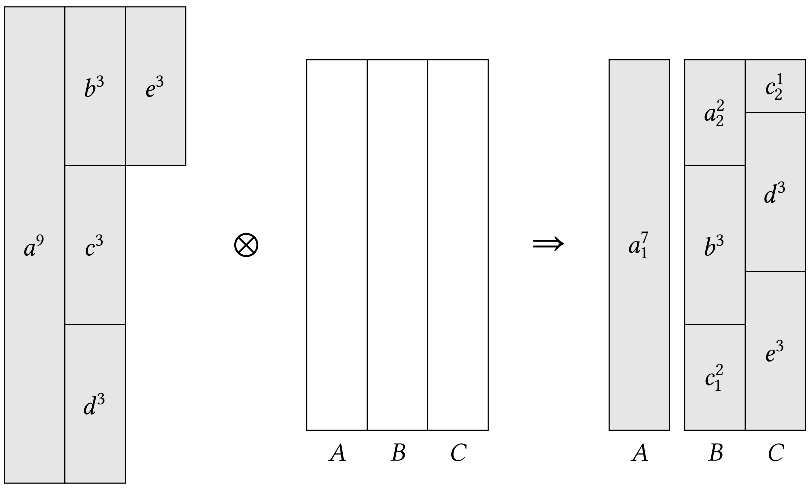


Figure 2.2: *Packing items into three bins of size seven allowing fragmentation.*

problem where items may be cut into smaller fragments. The objective is to minimize the number of item fragments, or equivalently, the number of cuts, for a given number of bins. Models based on fragmentable items are useful for representing finite, shared resources. Figure 2.2 shows an example problem instance.

Computational complexity involves algorithms that solve problems. The complexity of an algorithm is the amount of resources required to run it, while the complexity of a problem is the minimum complexity of all possible algorithms for that problem.

Because the resource requirements vary with the size of the input n, the complexity of an algorithm is expressed as a function $f(n)$, representing either worst-case or average-case complexity. This thesis only considers worst-case complexity, because it provides true guarantees. Computational complexity is commonly expressed in asymptotic notation, which describes the limiting behavior of a function when the argument tends towards infinity. This entails ignoring constants and low-order terms. In computer science, three common notations are Big O $\mathcal{O}(g(n))$, Big Theta $\Theta(g(n))$ and Big Omega $\Omega(g(n))$, denoting an upper bound, a tight bound and a lower bound, respectively.

The primary resource of interest is the time required to run an algorithm, this is known as time complexity and is the default. The secondary resource of interest is the amount of memory required to run an algorithm, which is known as space complexity. A classic, physical computer corresponds to a deterministic Turing machine. For such a machine, time complexity is a subset of space complexity. Hence, I only consider time complexity.

The complexity class P contains all decision problems that can be solved by a deterministic Turing machine in polynomial time, while the complexity class NP contains all decision problems that can be solved by a non-deterministic Turing machine in polynomial time, or equivalently, the decision problems where positive answers are verifiable in polynomial time. The hardest problems in NP are known as NP-complete. These are problems to which every other problem in NP can be reduced in polynomial time. A famous unsolved problem in computer science is whether P = NP, i.e. whether polynomial time algorithms exist for solving NP-complete problems. I have not attempted to solve this problem.

Optimization problems are related to decision problems in that an algorithm that exactly solves an optimization problem directly solves the corresponding decision problem. Every optimization problem is thus at least as hard as its corresponding decision problem. Most of the optimization problems studied in this thesis correspond to decision problems that have been proven NP-complete.

The classic bin packing problem is a well-known combinatorial optimization problem. Its decision form: "Can these items be packed into m bins?", is strongly NP-complete. This problem is a special case of bin packing with fragmentable items, i.e. "Can these items be packed into m bins with $k = 0$ cuts?", which makes the latter at least as hard as the former. Hence, any deterministic, exact algorithm requires time superpolynomial in the size of input, unless $\mathscr{P}$ = NP.

This paper addresses RQ5 from subsection 1.6.5. It adapts a metaheuristic known as a grouping genetic algorithm, developed by Quiroz Castellanos et al. (2015) for classic bin packing (BP), to the problem of minimum fragmentable items bin packing (MIN-FIBP), presented by LeCun et al. (2015). By reducing the complexity of the approximation algorithms for MIN-FIBP developed by LeCun et al. (2015), I succeed in turning them into fast packing heuristics for the grouping genetic algorithm while maintaining their approximation guarantees. I develop fast lower bounds for the problem, to enable early termination through identification of optimal solutions. The lower bounds and approximation algorithms provide approximation guarantees known as performance ratios. The worst-case performance of a lower bound L for a problem P, where OPT(I) is the optimal value of P for instance I, is given by:

$$r(L) \triangleq \sup\left\{ \frac{L(I)}{\mathrm{OPT}(I)} \,\middle|\, I \text{ is an instance of } P \right\}$$

and the asymptotic worst-case performance of L is given by

$$r(L)_\infty \triangleq \limsup_{s\to\infty} \left\{ \frac{L(I)}{\mathrm{OPT}(I)} \,\middle|\, I \text{ is an instance of } P \text{ with } \mathrm{OPT}(I) \geq s \right\}.$$

The asymptotic worst case performance is the value to which the maximum tends in the limit, as shown in Figure 2.3, where the maximum tends to 1. Note that the worst-case performance here is 2.

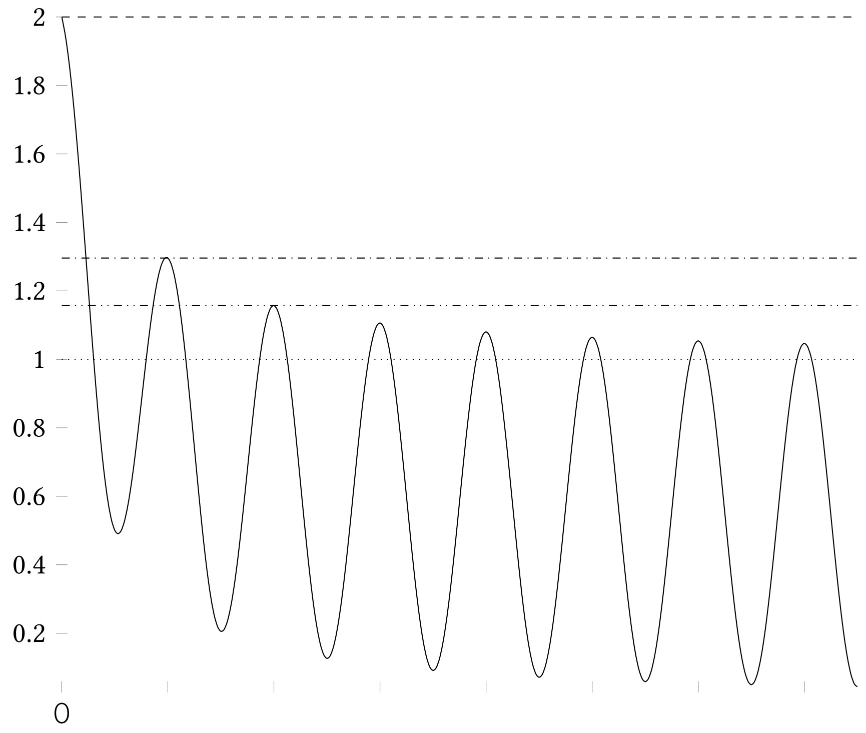


Figure 2.3: *The difference between worst-case performance and asymptotic worst-case performance is that worst-case performance is the extreme value over the entire range,* 2 *in this illustration, whereas the asymptotic worst-case performance is the value that the extreme value tends towards in the limit,* 1 *in this case.*

Algorithm 1 *This algorithm yields all possible 3-partitions of a number r from a set of m numbers in the sorted array a.*

```
procedure THREESUM($(a, m, r)$)
    $n \leftarrow 0$
    if $r = 0 \vee n = m$ then
        return
    end if
    $m \leftarrow m - 1$
    while $a[m] + 2a[n] < r$ do
        $m \leftarrow m - 1$
        if $m < n$ then
            return
        end if
    end while
    while $n \leq m$ do
        while $a[n] + 2a[m] > r$ do
            $n \leftarrow n + 1$
            if $n > m$ then
                return
            end if
        end while
        $v \leftarrow n$
        $w \leftarrow m$
        $s \leftarrow r - a[m]$
        repeat
            $t \leftarrow a[v] + a[w]$
            if $t < s$ then
                $w \leftarrow w - 1$
            else
                if $t = s$ then
                    yield $(v, w, m)$
                end if
                $v \leftarrow v + 1$
            end if
        until $v > w$
        $m \leftarrow m - 1$
    end while
end procedure
```

I present a fast grouping genetic algorithm for the bin packing problem with fragmentable items. It delivers a worst-case guarantee of a 5/4-approximation with complexity $\mathcal{O}(n \log n + |\mathscr{W}|^2) = \mathcal{O}(n^2)$. For any given problem instance, $|\mathscr{W}|$ is a constant, far smaller than n. This also holds for many practical applications, despite varying the number of items n. Under these circumstances, the complexity is only $\mathcal{O}(n \log n)$, which is especially attractive when used as a frequently evaluated packing heuristic.

I study the performance of the involved algorithms through extensive computational experiments involving two problems sets: $\mathscr{P}_1$ consisting of 180 problem instances and $\mathscr{P}_2$ consisting of 450 problem instances. Since the grouping genetic algorithm is nondeterministic, I solve each problem instance 10 times, to get a better understanding of how the algorithm usually performs.

The metaheuristic works well in practice. With problem set $\mathscr{P}_1$, it reached the optimum in 92 % of runs (1662 out of 1800 runs). No run took longer than 17 ms. Most of the 1800 runs actually completed in less than 6 ms. With problem set $\mathscr{P}_2$, it reached the optimum in 99 % of runs (4473 out of 4500 runs). No run took longer than 220 ms. Most of the 4500 runs completed in less than 13 ms.

To this end, I made the following contributions: 1. I developed the greedy algorithm $\mathscr{G}^+$ for constructing a feasible solution. It has worst-case complexity $\mathcal{O}(n)$ and works for general instances of MIN-FIBP. 2. I adapted the exact algorithm $\mathscr{E}_2$ to the general case of minimum fragmentable items bin packing with equal capacities (MIN-FIBP-EQ). This avoids the need for a pseudo-polynomial transformation. 3. Algorithm $\mathscr{B}_3$ is a 1/3-approximation with complexity $\mathcal{O}(n + |\mathscr{W}|^2) = \mathcal{O}(n^2)$ for finding blocks of size 3 in the general case of MIN-FIBP-EQ. Its predecessor $\mathscr{A}_3$ is a 1/3-approximation with complexity $\mathcal{O}(n^4)$ for the special case of MIN-FIBP-EQ, where the sum of item sizes is equal to the aggregated bin capacity. The key behind algorithm $\mathscr{B}_3$ is the fast computation of 3-partitions through Algorithm 1 and subsequent selection between feasible partitions. 4. I designed a new family of fast lower bounds for MIN-FIBP-EQ and proved their worst-case performance ratios. The best lower bound offers a guaranteed worst-case performance of 3/4. In practice, it usually performs far better than this. With the first problem set, it performed no worse than 5/6. With the second problem set, it performed no worse than 9/10. Its worst-case complexity is $\mathcal{O}(n \log n)$. 5. I made significant changes to a grouping genetic algorithm for classic bin packing, as well as its constituent operators, and adapted it to the new problem MIN-FIBP-EQ. I also proposed a more efficient genome representation for all grouping genetic algorithms designed for bin packing. 6. I created a reference implementation of a state-of-the-art solver incorporating the proposed algorithms and made it available to the public. 7. I designed a comprehensive problem instance set $\mathscr{P}_2$ to evaluate the solver and made it publicly available. I published the full result set to allow for future comparisons.

2.6 Optimized Deployment Plans for Platform as a Service Clouds

I approximately solve the NP-hard problem of computing deployment plans for multiple cloud services by presenting approximate and heuristic algorithms that can operate under soft real-time constraints. The objective is to find an optimal assignment of services to servers, minimizing the number of servers and service instances, as service instances incur overhead through memory use and complex management, while running servers cost money and energy. I also wish to do this quickly, due to the high volatility of many Internet-scale applications.

I target a PaaS utility model where the cloud provider offers a computing platform with automatic resource management. This is in contrast to an IaaS model, where the basic offering is the VM. In the IaaS model, a VM must not require more computing resources than what is provided by a PM. In both models, the cloud provider allocates multiple VMs to a single PM and tries to keep its utilization within allowable limits in order to leverage its hardware investment.

The distinction between IaaS and PaaS is somewhat blurred, but the traditional view of PaaS is that it provides users access to elastic computing resources without the need for explicit management. The PaaS model suits services that can be deployed in multiple VMs, resulting in increased reliability and performance, since the performance and reliability requirements of a service may exceed that which is offered by a single physical server.

The platform in PaaS can be simply regarded as middleware, which any IaaS user also can leverage to deploy a service in multiple VMs and then use a scaling and load balancing mechanism to distribute the work among dynamically allocated VMs. Indeed, many scholars have studied the problem of horizontal scaling from the perspective of the IaaS user. A common theme in these works is that cloud providers leverage existing IaaS clouds to deploy services in combination with an auto-scaling mechanism for deciding how many VMs a given service will require at any given moment, either now or in the future, as well as a load balancer for distributing the load between these VMs. In this way, cloud providers obtain the benefits of a PaaS cloud while using a basic IaaS cloud, giving greater control to the downstream provider. However, from a holistic perspective, it is actually beneficial to leave the responsibility of elastic scaling and resource management to the upstream provider, which can optimize the entire system globally.

For example, assume that a data center hosts three services (a, b and c), from three different users in three different servers, and that each server is capable of handling 500 million instructions per second (MIPS). The demand is 600 MIPS for service *a*, 100 MIPS for service *b* and 300 MIPS for service *c*. Since the resource demand for service *a* exceeds the capacity of a server, the downstream provider deploys the service in two VMs: a_1 and a_2, requiring 300 MIPS each (horizontal scaling), using a load-balancing mechanism to spread the requests evenly between the VMs. Now each server is underutilized, partially due to the policy that the load associated with service *a* is evenly distributed among its two instances. The downstream provider cannot know that this situation is suboptimal. However, taking into account the computing needs of all services in the data center, one can find a better allocation of VMs to servers and consolidate the services in only two servers. This optimal solution cannot be obtained if the upstream provider is bound by the decisions made by the downstream provider. Only the upstream provider knows the complete picture, so by leaving resource management to the upstream provider, a better outcome is achievable. This leads to lower operational costs for the upstream provider and a lower environmental footprint for all involved entities. The upstream provider can in turn reduce its prices, gaining an edge over its competitors, also benefiting the downstream provider.

Due to the high volatility in the load of many Internet-scale applications, the algorithms involved in deployment planning must be fast and produce good solutions before they become irrelevant due to changed premises. That is, they must operate under soft real-time constraints. The algorithms must cope with large clusters consisting of millions of services and servers without spending too much resources on producing the deployment plans, since they are merely means to an end and do not possess any intrinsic value.

Resources are discrete quantities. To make the problem tractable, resource supply and demand must be quantized at some resolution, e.g. in a scenario where CPU is the bottleneck resource and the most powerful available server offers 500 MIPS, quantizing supply and demand as units of 100 MIPS yields a set of 5 distinct resource requirements for the entire system. By definition, any service requiring more than 500 MIPS must be broken up into distinct parts requiring no more than 500 MIPS each.

Several authors have used classic BP as the underlying model for such algorithms. However, algorithms based on classic BP must always produce deployment plans using unaltered services with fixed size, since they cannot alter the sizes of the VMs that will be deployed. This greatly limits the applicability of the classic BP model.

For this reason, I have devised an approach based on fragmentable items bin packing (FIBP), which is a generalization of classic BP and can produce variable-size deployment plans. This model enables full utilization of an optimal number of servers, as opposed to a classic BP model, which leaves many

Table 2.1: *Example of deployment plans*

(a) *Fixed-size plan*

	Server		
Service	S_1	S_2	S_3
a	3/5	3/5	
b	1/5		
c			3/5

(b) *Variable-size plan*

	Server	
Service	S_1	S_2
a	5/5	1/5
b		1/5
c		3/5

servers underutilized. It naturally incorporates services too large to fit in any one server, since these require splitting, and is approximately solvable under soft real-time constraints, where the goal is minimizing the number of service fragments.

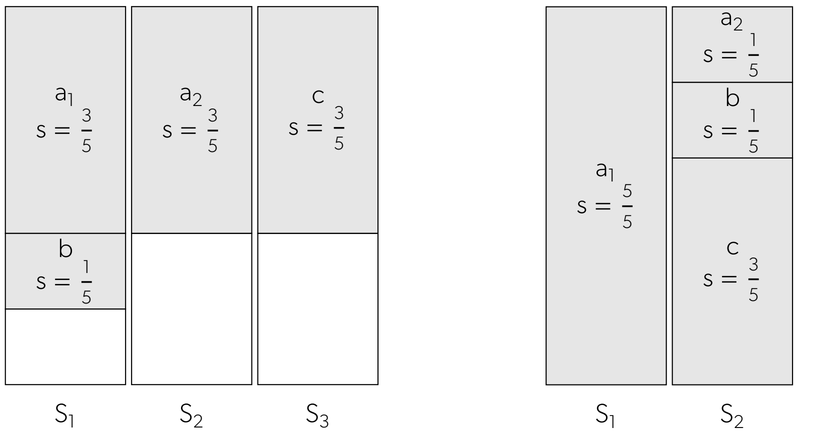
The optimal number of servers is attained practically for free, since it is straightforward to compute through the linear relaxation of the packing problem, and any valid FIBP solution will satisfy this optimal set of servers. Thus, it is a further benefit of the FIBP model that its objective merely consists of reducing the number of service fragments, not in minimizing the number of servers. The optimal number of servers for a fragmentable items bin packing with equal capacities (FIBP-EQ) problem is given by Equation 2.1, where the sum of item sizes $\sum s$ is divided by the unique bin capacity c.

$$L_1 = \left\lceil \frac{\sum s}{c} \right\rceil \tag{2.1}$$

Figure 2.4 illustrates how a fixed-size plan, such as one produced by solving a classic BP problem, is less efficient than a variable-size plan, such as one obtained from MIN-FIBP. In the example, there are three services: a, b and c, with respective capacity requirements of 6/5, 1/5 and 5/5. Each server provides 5/5 units of processing power. A classic BP model is not directly applicable to this problem, since service a does not fit in any one server.

If one modifies the problem and splits service a in two equal pieces, a_1 and a_2, each requiring 3/5 of a server, as shown in Table 2.1a, one can solve the problem using 3 servers, but each server is underutilized, as shown in Figure 2.4a. If one instead were to split service a in two unequal pieces: a_1 with capacity requirement 5/5 and a_2 with capacity requirement 1/5, as shown in Table 2.1b, one can fit all services in two servers with full utilization, as shown in Figure 2.4b.

In the classic BP problem n items (services) and n bins (servers) are given. The goal is to produce an assignment X of items to bins with a minimum number of bins in use. Each bin has a capacity c, which may not be exceeded. Each item i has a size s_i requiring some capacity of the bin in which it is placed.



(a) *Fixed-size services.* (b) *Variable-size services.*

Figure 2.4: *Example of service deployment.*

The classic BP problem can be formalized as

$$\begin{aligned}
\min_{z} \quad & z = \sum_{i=1}^{n} y_i && (2.2a)\\
\text{subject to} \quad & \sum_{j=1}^{n} w_j \chi_{ij} \le c y_i, \ \forall i \in \mathbf{N} = \{1, \dots, n\}, && (2.2b)\\
& \sum_{i=1}^{n} \chi_{ij} = 1, \ \forall j \in \mathbf{N}, && (2.2c)\\
& y_i \in \mathbb{B}, \ \forall i \in \mathbf{N}, && (2.2d)\\
& \chi_{ij} \in \mathbb{B}, \ \forall i, j \in \mathbf{N}. && (2.2e)
\end{aligned}$$

This paper applies the fragmentable bin packing model from Byholm and Porres (2018) to cloud computing and shows why the model is a good fit. It is effectively serverless computing. The paper thus addresses RQ5 from subsection 1.6.5. I present a formal definition of the problem of deployment planning for cloud services in a PaaS context, as well as planning algorithms based on FIBP that work under soft real-time constraints.

Min-FIBP is the optimization version of FIBP. In Min-FIBP, the minimum number of servers required to solve a problem is always known. The objective is to minimize the number of fragments for given sets of items and bins. A solution with no cuts is equivalent to a classic BP solution. The Min-FIBP problem can be formalized as follows: It comprises an assigment X of services $s \in \mathrm{S}$ to servers $m \in \mathbf{M}$. The objective Equation 2.3a is to minimize the number of fragments, i.e. integer entries $\chi_{sm} > 0$. The first constraint Equation 2.3b is that each service $s \in \mathrm{S}$ must be fully assigned to some servers $m \in \mathbf{M}$, so the sizes of all containers of service s must add up to the quanta $\mathrm{r}(s)$. The second constraint Equation 2.3c states that each server $m \in \mathbf{M}$ has a capacity $\mathrm{R}(m)$, which must not be exceeded. The third constraint Equation 2.3d restricts containers to multiples of unit-sized parts, so the size of each container must be a natural number:

$$\min_{X} \quad \sum \mathbb{1}_{X>\mathbb{0}} \tag{2.3a}$$

$$\text{subject to} \quad \sum_{m \in \mathbf{M}} \chi_{sm} = \mathrm{r}(s), \ \forall s \in \mathbf{S}, \tag{2.3b}$$

$$\sum_{s \in \mathbf{S}} \chi_{sm} \leq \mathrm{R}(m), \ \forall m \in \mathbf{M}, \tag{2.3c}$$

$$\chi_{sm} \in \mathbb{N}, \ \forall (s, m) \in \mathbf{S} \times \mathbf{M}. \tag{2.3d}$$

I approximately solve the MIN-FIBP problem using the grouping genetic algorithm from Byholm and Porres (2018). The algorithm yields a guaranteed 5/4-approximation with complexity $\mathcal{O}(|\mathbf{S}|^2)$ in the worst case. For actual applications, the resolution of resource requirements is a small constant, much smaller than the number of services $|\mathbf{S}|$. In every case like this, the worst case complexity is only $\mathcal{O}(|\mathbf{S}| \log |\mathbf{S}|)$.

Through computational experiments involving 10 random deployment planning problems with 256 services each, I show how my algorithm produces optimal deployment plans for hundreds of servers in about 3 milliseconds. This is contrasted with attempting to solve the same models using the well-known, award-winning, state-of-the-art solver CPLEX for mathematical programming, which produces 30 % worse results, given 100 seconds per problem. I also compare the classic BP model to the FIBP model and show that the FIBP model requires 2–4 fewer servers than the BP model for every studied problem. I thereby show that FIBP is a better model than classic BP for the problem of computing deployment plans for multiple cloud services in soft real-time.

2.7 Dynamic Horizontal and Vertical Scaling of Multiple Cloud Services in Soft Real-Time

I approximately solve the NP-hard problem of continuous deployment and scaling for multiple cloud functions in a serverless computing setting. In this problem, a number of users want to deploy multiple computing functions in the cloud. Each function may state a minimum CPU capacity reservation, in order to ensure a suitable response time. However, the CPU capacity reservation may be larger than that provided by a physical server and the mapping of functions to containers and servers is up to the provider.

I solve this problem by presenting approximate and heuristic algorithms that can operate under soft real-time constraints. The objective is to find an optimal assignment of functions to servers, minimizing the number of servers and containers, as containers incur overhead through memory use, while running servers cost money and energy. Figure 1.2 depicts the problem domain. Sessions can move between containers implementing a service at a cost. Each session belongs to a specific, fixed service and has an affinity for a specific container that varies over time. This affinity captures the behavior of storing session state locally in containers, which might lead to better performance, as documented in Byholm and Porres (2014).

I study the elastic version of this problem, where users may vary the CPU reservation of a function over time. In this case, the cloud platform must not only optimize the number of running servers, but also the number of function configuration management operations, such as deploying, migrating or removing containers from servers, since each one of these management operations incur CPU and traffic overheads.

Serverless computing is a fairly new concept in academia. It is a variant of classic PaaS but with less control for users. This reduced control is not necessarily negative, since users are freed from the responsibility of managing the deployment. Less control for users means more control for providers. Providers can optimize the deployment more efficiently, leading to reduced costs.

The FaaS model suits functions that can be deployed in multiple servers, resulting in increased reliability and performance, since the performance and reliability requirements of a function may exceed that offered by a physical server.

This paper addresses RQ6 from subsection 1.6.6 by combining the deployment planner from Byholm and Porres (2017b) with an algorithm for transitioning between deployments. It relates to Ashraf, Byholm, and Porres (2016), Ashraf, Byholm, and Porres (2015) and Byholm and Porres (2014). I introduce the concept of elasticity to the model and provide algorithms for continuous migration and dynamic updating. The main contribution of this article are: a) A formal definition of the problem of deployment planning for cloud functions in a FaaS context, b) deployment planning algorithms based on FIBP that work in a dynamic setting and c) a migration algorithm for transitioning between deployments at low cost.

The main algorithm executes whenever the sets of sessions for each service changes. If a new session arrives, there are two possibilities: Either there is sufficient capacity in some container, and the session is assigned to that container, or no container can accommodate the session. If no container has sufficient capacity, at least one container must grow by at least one unit. If there is a suitable container on a server with unused capacity, one simply grows that container by one unit and assigns the session there. If no container has room to grow, one adds a server, makes a new assignment plan and finally one migrates sessions and containers to realize the new plan. If a session leaves, its container is shrunk, if possible. If this reduction in container size allows using at least one server fewer, one makes a new assignment plan, migrates sessions and terminates the then unused servers.

I represent the problem of assigning services to containers and containers to services as the MIN-FIBP problem, like I did in Byholm and Porres (2017b). I solve the problem in $\mathcal{O}(|\mathrm{S}|^2)$ using the method of Byholm and Porres (2018). For real-world problems, this complexity is further reduced to $\mathcal{O}(|\mathrm{S}| \log|\mathrm{S}|)$. This produces an abstract plan of containers with various sizes assigned to servers. One must now make a concrete realization of this plan by mapping existing containers to containers in the new plan and migrating sessions. I reduce the problem of migrating sessions among containers to a linear assignment problem, which I solve using an implementation of the LAPMOD algorithm by Volgenant (1996). I solve the assignment problem as follows: Given an existing deployment $\boldsymbol{A}$, a desired deployment $\boldsymbol{B}$ of equal size and a cost matrix $\boldsymbol{\Phi}$, find an assignment $\boldsymbol{\Omega} : \boldsymbol{A} \rightarrow \boldsymbol{B}$ with minimum cost.

To compute the cost of an assignment, one forms the cost matrix $\boldsymbol{\Phi}$ as the amount of data (including code and cached program resources and application state) that must be moved to make $\boldsymbol{A}$ and $\boldsymbol{B}$ equivalent. All data that are moved out from a server must be moved in to another server. This means that one only needs to compute the cost of moving data out:

$$\boldsymbol{\Phi} = [\sum\{\mathrm{d}(c) : c \in \bigcup \alpha \ominus \beta\} : (\alpha, \beta) \in \boldsymbol{A}_{\mathrm{M}} \times \boldsymbol{B}_{\mathrm{M}}], \tag{2.4}$$

where $\mathrm{d}(c)$ gives the size of data in (part of) a container c, while $\alpha \ominus \beta$ denotes a form of set difference between sets $\alpha \subseteq \boldsymbol{A}_{\mathrm{C}}$ and $\beta \subseteq \boldsymbol{B}_{\mathrm{C}}$, where a container in α that is also found in β is reduced to a container with capacity for a subset of the smallest function invocations that must be moved out, given the capacity of the container from β, while containers in α that are not found in β are returned as usual. In other words: if a container for a function is found in both the existing and the desired assignment and the container in the desired assignment is smaller, return the smallest function invocations that must be moved to meet the new capacity requirements, otherwise apply regular set difference. I then solve the ensuing problem:

$$\min_{\boldsymbol{\Omega}} \quad \langle \boldsymbol{\Phi}, \boldsymbol{\Omega} \rangle_{\mathrm{F}} \tag{2.5a}$$

$$\text{subject to} \quad \sum_{\alpha \in \boldsymbol{A}} \omega_{\alpha\beta} = 1, \ \forall \beta \in \boldsymbol{B}_{\mathrm{M}}, \tag{2.5b}$$

$$\sum_{\beta \in \boldsymbol{B}} \omega_{\alpha\beta} = 1, \ \forall \alpha \in \boldsymbol{A}_{\mathrm{M}}, \tag{2.5c}$$

$$\omega_{\alpha\beta} \in \mathbb{B}, \ \forall (\alpha, \beta) \in \boldsymbol{A}_{\mathrm{M}} \times \boldsymbol{B}_{\mathrm{M}}, \tag{2.5d}$$

where $\langle \boldsymbol{\Phi}, \boldsymbol{\Omega} \rangle_{\mathrm{F}}$ is the Frobenius inner product of $\boldsymbol{\Phi}$ and $\boldsymbol{\Omega}$.

I present a new take on the NP-hard problem of continuous deployment and scaling for multiple cloud functions with soft real-time constraints. I recognized that this is a generalized bin packing problem with fragmentable items combined with an assignment problem. I formalized the problem domain and designed scaling algorithms for a FaaS platform. I recognized that the FIBP model is a better fit than the classic BP model, since it enables using fewer servers, supports heterogeneous servers with multiple configurations and copes with functions too large to fit in any single server. The FIBP model enables denser packings with less wasted capacity. The linear assignment problem offers an efficient way of transitioning between deployments. While theoretically sound, the proposed design has yet to be implemented, integrated with the ARVUE system and verified experimentally. The work remains at an early stage and an extension would be an excellent research contribution.

2.8 A Multi-Objective ACS Algorithm to Optimize Cost, Performance, and Reliability in the Cloud

Public cloud providers such as Amazon, Google and Microsoft operate large data centers around the world using economies of scale to offer computing resources at competitive prices. Their IaaS clouds offer various types of VMs with varying levels of cost, performance and reliability. Some VM types have large amounts of RAM relative to CPU, others provide more resources in all aspects and some have specialized hardware, such as graphics processing units (GPUs) for general-purpose computing on graphics processing units (GPGPU). The choice of VM types and how many VMs of each type to procure directly affects the QoS and cost of a system. A large number of contemporary software systems are built from software components that can be deployed on one or more VMs. A typical software system comprises a number of software components, where each component often requires certain levels of performance and reliability. Therefore, when deploying a component-based software system in an IaaS cloud, performance and reliability requirements of individual software components should be taken into account. In practice, it is often possible to provide high QoS levels by over-provisioning resources. However, over-provisioning results in increased cost of operation. Therefore, performance and reliability of software systems can not be optimized in isolation from the resource cost. Thus, a software deployment configuration should be simultaneously optimized in terms of cost, performance and reliability.

The cloud-based software component deployment problem is a special case of the generic software architecture optimization problem, in which the search-space of architecture design alternatives is explored with respect to one or more objectives. The component-based software development paradigm provides various generic architectural degrees of freedom that can be exploited to create different functionally equivalent alternatives of an architectural design. An architectural degree of freedom refers to a way an architecture model can be modified and improved in terms of certain quality properties without affecting the functionality of the system. For instance, the component allocation degree of freedom allows changing the allocation of software components to servers in order to optimize a software architecture model with respect to certain objectives. Thus, architectural degrees of freedom define the search-space for optimization in which all solutions provide the same functionality, but with different qualitative properties.

This paper defines the cloud-based software component deployment problem as a multi-objective optimization problem with three antagonistic objectives: 1. cost, 2. performance and 3. reliability. Manually exploring the search-space of deployment configurations with respect to three antagonistic objectives is time-consuming, error-prone, and may lead to sub-optimal solutions. Moreover, since the multi-objective cloud-based software component deployment problem is an NP-hard combinatorial optimization problem, it should be approached in a systematic way by using efficient optimization techniques. Furthermore, since the problem involves multiple objectives, single-objective optimization techniques are not appropriate. Therefore, it should be approached with a multi-objective optimization technique that produces a set of Pareto-optimal configurations, which can later be evaluated on subjective criteria. It is up to a utility function to pick the most preferred one, since this cannot be done in an objective manner.

Pareto-optimal solutions to a multi-objective optimization problem lie on the Pareto frontier. Figure 2.5 illustrates a Pareto frontier. Here, solution C is dominated by solutions A and B, since both A and B score lower than C in both objectives f_1 and f_2. Solutions A and B do not, however, dominate each other, since A is better than B in objective f_2, but worse in objective f_1.

This paper presents an ant colony system approach for multi-objective optimization and compares it with a basic implementation of the well-known genetic algorithm NSGA-II by Deb et al. (2002). It considers three degrees of freedom: component allocation, VM selection and number of VMs. The ant colony system outperforms the genetic algorithm in terms of quality and number of Pareto-optimal solutions found, but its implementation is far slower. This inefficiency could likely be improved in a production implementation. In contrast to many existing approaches, the search algorithms function without requiring initial, user-supplied architecture configurations, since that might limit the search to a local area around the initial starting point. The method is intended for design-space exploration rather than micro-optimization of a given design. The paper addresses RQ7 from subsection 1.6.7 by exploring design spaces through multi-objective optimization and Pareto analysis. The

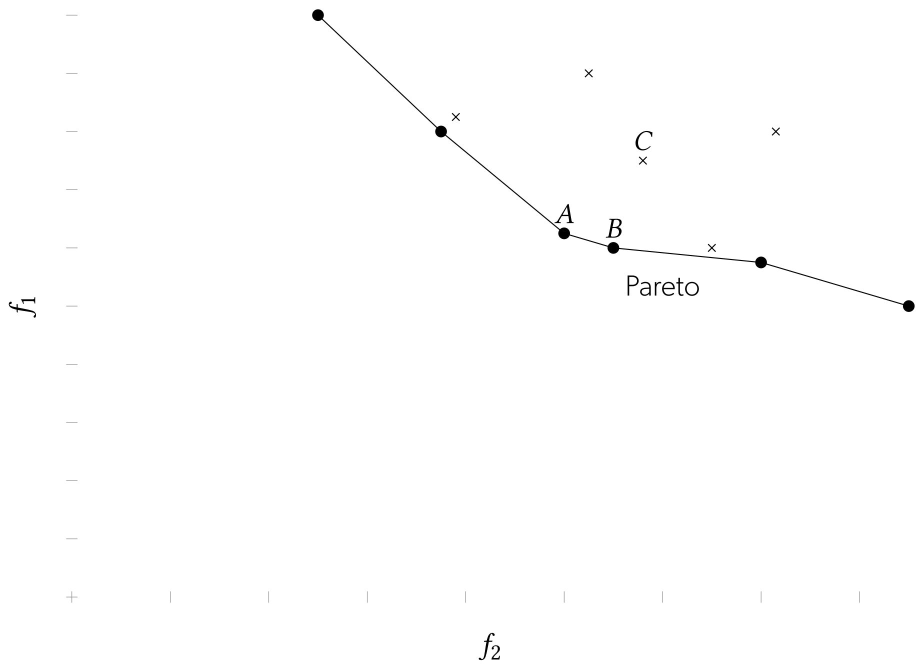


Figure 2.5: *The Pareto frontier forms a Pareto set consisting of non-dominated solutions. In this multi-objective minimization problem, solution C is dominated by solutions A and B, since both A and B score less than C in both objectives f_1 and f_2. Solutions A and B do not, however, dominate each other, since A is better than B in objective f_2, but worse in objective f_1.*

approach has been experimentally validated through a test problem loosely based on an actual production system used in industry. The ant colony system algorithm is compared with an alternative implementation using the well-known NSGA-II algorithm by Deb et al. (2002). While the implementation of the ant colony system algorithm is vastly slower than the alternative based on NSGA-II, it does produce more solutions of higher quality for this specific case study. One should also bear in mind that it is an early research prototype focused on correctness, not on speed, whereas the NSGA-II implementation is production ready. Neither approach is currently suitable for use as a frequently evaluated routine in a time-sensitive cloud system, but further research into that area is warranted.

The research was done in close collaboration with Adnan Ashraf, who designed the ant colony system algorithm. I designed the problems and experiments, analyzed the data, drew figures and developed the alternative approach based on NSGA-II. I also aided in development of the scoring functions of the ant colony system algorithm.

2.9 Discussion

I propose a new take on stateful serverless computing, defining its boundaries while answering open research questions on the essence of serverless computing. Serverless computing is not inherently stateless. In fact, no serious computing is stateless. My work has a solid basis in proven theory, in addition to experimental verification. The presented papers form the basis of what is required of a modern, cost-efficient, autonomous, serverless computing platform. Stateful serverless computing is no silver bullet, but neither is anything else. It is not sensibly applicable to every conceivable use case within computing in general, nor cloud computing in particular. Stateful serverless computing works when there are services associated with sessions in an OLTP setting.

When designing the system, I have considered three primary quality factors: time, cost and reliability. As usual, the objectives are mostly antagonistic and two of the options may be chosen at the expense of the third.

Machines are meant to serve humans. Hence, at the end of every computation sits a human waiting for a result. Humans have finite life spans, which is why time is of the essence and latency should be minimized. This is especially important in an OLTP context, which is what I primarily consider, as opposed to batch processing. I have taken great care when designing the system to minimize latency wherever possible.

Cost is important to the extent that service providers should be profitable, else there would be no service providers, but time and quality must not suffer too much at the expense of cost, since that would likely lose customers. Reducing latency and maintaining reliability keeps users content, which is beneficial to all involved parties. Increasing efficiency also reduces cost for service providers, further increasing their profit margins.

The system should be reliable enough to maintain customers, but must still meet the time and cost restrictions. Always ensuring perfect reliability for every system interaction would often be too slow or expensive. Separating data into soft and hard state, as I propose in this thesis, allows having lighter reliability guarantees on soft state, which helps reduce latency and lower costs.

Because this work and the entire field of stateful serverless computing is in its infancy, there are many relevant, concrete implementation details which have not been studied in detail. Again, I do not believe that any single implementation of the described system would be ideal for every possible use case. While stateful services with HTTP sessions constitute the recurring, practical use case throughout this thesis, this is not the only straightforward application. Yet, there is a fine balance between specificity and generality. Describing something too generically can lead to obfuscation, while being overly specific hides the larger picture. In the end, successful products tend to be more specific than general. To compare with many-objective optimization and Pareto efficiency, pick one or two primary objectives and optimize for them, because it is not possible to satisfy them all. As the old adage tells: Jack of all trades, master of none.

To contrast my work from FaaS, I have not envisioned a swarm of tiny functions composed to make a service, since that already exists and has several known drawbacks, as previously discussed. For lack of a better name, my vision could be described as service as a service, keeping data local to where it is needed, preserving state over a long duration, increasing cohesion and lowering coupling. This model is closer to traditional, monolithic design, which reduces latency. At the same time, there should be enough flexibility to easily restructure service deployments and manipulate the server pool in an autonomous fashion so that the system can adjust to variations in demand and provide redundancy to increase availability. I will conclude this discussion by reflecting back on the research questions posed in section 1.6 and answering them in a few sentences each.

The generic answer to RQ1 from subsection 1.6.1 on how to build an autonomous serverless computing platform is to treat services as black boxes, measure their impact on key performance indicators of the whole system and take appropriate scaling decisions based on these measurements. The answer to RQ2 in subsection 1.6.2 on how to optimize VM consolidation is to pick a model which can be solved quickly enough for a system of a particular size, optimizing for the cost of consolidating the servers compared to the cost of doing nothing, while accounting for the estimated future demand. RQ3 in subsection 1.6.3 asked how to optimize session management. The answer is to treat session state as soft state, maintain the affinity for sessions to particular servers, cache session state locally in the application server and settle for eventual consistency by asynchronously storing session state in a reliable store. RQ4 in subsection 1.6.3 asked how to optimize caching. The answer is that there is a constant storage duration obtained by dividing the cost of computing an answer by the cost of storing said answer over time. Only store data if at least one new request is likely to arrive for the same data during the subsequent storage period. The anwer to RQ5 in subsection 1.6.5 on which optimization model suits serverless computing. The answer is bin packing with fragmentable items, since it is better than classic bin packing in every aspect. RQ6 in subsection 1.6.6 asked how to optimize scaling of stateful serverless systems. My answer is to quickly compute a new desired deployment configuration, determine whether a transition to the new new deployment configuration is worth the effort and only then to transition into the new deployment configuration through the shortest, cheapest path. As for RQ7 in subsection 1.6.7 on how to optimize service deployments with many objectives, the answer is not to do it, avoiding this scenario at all cost, because true many-objective problems are too hard and complex to effectively solve in any reasonable amount of time. Take every opportunity to discard insignificant objectives, combine objectives using aggregate objective functions and redefine the problem until it no longer is a true many-objective problem. Many-objective problems warrant significant future research, but it might well be a dead end.

And programming computers was so fascinating. You create your own little universe, and then it does what you tell it to do.

Vint Cerf

3 Conclusion

This thesis presents resource management algorithms for many different problems in the field of serverless computing, as well as general cloud computing. Contrary to the FaaS paradigm, which might be the most well known form of serverless computing, I have incorporated the management of state into the serverless system.

The ARVUE platform presented in Paper I was initially developed by Ashraf, Byholm, Lehtinen, et al. (2012), before the term serverless computing existed, and possesses all the necessary features to be considered a stateful serverless computing platform. The platform seamlessly manages the computational resources required for hosting stateful, third-party applications in a cost-efficient manner, without charging for idle time.

VM consolidation aims to optimize the placement of VMs in PMs. Paper II studied existing approaches to VM consolidation for large-scale, distributed systems, comparing and categorizing various solutions to this problem.

Optimization problems involving more than one objective do not produce a single optimal solution, since there are trade-offs between the antagonistic objectives. Instead, they produce a set of Pareto-optimal solutions, none strictly worse than any others. Paper VIII studied the problem of deploying services requiring multiple resources to computation nodes with various cost, performance and availability guarantees.

Many systems involve stateful services with sessions, where each user belongs to an active session, containing state information, such as access credentials, profile data and work in progress. This state must be kept somewhere available to the application server handling a particular request. Storing state locally in RAM at an application server is fast, but memory is limited and volatile. A fault anywhere in the application server can easily lead to loss of the stored state, which will be perceived as a failure. Load balancing is also harder, since all requests belonging to a session must be handled by the server containing said state. Storing session state in a reliable, remote database, as is common when a so-called stateless service requires state, adds cost and latency due to data transfers. In Paper III, I studied various approaches to optimized session management.

There are computations that require substantial computational effort and produce large amounts of data. Paper IV developed an autonomous system that balances the cost of recomputing a data set versus storing it for a given duration.

One of the strengths of serverless computing is that the cloud provider is free to manage the allocation of computing resources without input from the users. In this context, the allocation of sessions, services, containers and servers may be represented as a new abstract optimization problem known as MIN-FIBP. In Paper V, I presented fast algorithms for solving said problem, which I applied in Paper VI after studying the merits of FIBP over BP. In Paper VII, I extended this from the static deployment planning case to the dynamic reconfiguration case.

Figure 3.1 shows the combined platform management algorithm. Operating in conjunction with the underlying stateful serverless computing platform primarily developed in Paper I, shown in Figure 1.3, the prescribed algorithm executes whenever there is a change in the set of sessions. If a new session arrived, check for a suitable container implementing the service to which said session belongs. If a suitable container with enough remaining capacity exists, assign the session to that container. If no suitable container is available, the system must scale up to meet the increase in demand. First attempt vertical scaling by checking if an existing container has room to grow with its server. If such a container exists, grow the container and assign the session to said container. Otherwise, perform horizontal scaling by adding a new server. When doing horizontal scaling, it is a perfect opportunity to reduce fragmentation by creating a new deployment of services to servers, so solve a new MIN-FIBP instance and migrate the existing deployment to this new deployment with minimal effort, as described in Paper VII.

Before migrating between deployments, make sure to synchronize affected session state to the reliable store formed by the repository. The session manager described in Paper III is then able to successfully maintain the state for said sessions. Since the changes between deployments are minimized by the migration algorithm and the entire system is continuously kept close to an optimal configuration, the impact of a migration operation is kept minimial.

Conversely, if a session left the system, check if the system can scale down vertically by seeing its associated container can shrink without becoming overloaded. If that is the case, shrink the container and see if the system could scale down horizontally by removing a server. If this is possible, create a new service deployment with fewer servers than the existing deployment by again solving a new MIN-FIBP instance and migrating to this new deployment in the same manner as when scaling up. On the other hand, if the affected container could not be shrunk or a server could not be removed, do nothing.

3.1 Future Work

The stateful serverless computing platform presented in this thesis is still in its infancy. Hence, there remains plenty of practical future work in assembling all the described components into a functioning unit, tuning parameters for a specific use case, selecting which management policies to use and adding redundancy to components where desired before it can be used in production. Alas, building a fully functioning production-quality system of this magnitude requires more resources than I have at my disposal as a poor doctoral student and warrants proper compensation, which only the private sector has to offer.

There is, however, one remaining research avenue left to explore: A detailed simulation of the complete system at a concrete level, e.g. using CloudSim by Calheiros et al. (2011). Testing a feature-complete, fully functioning implementation in a simulator together with actual traces of relevant activity in a similar production cloud could yield new insights, as well as shorten the development process of a concrete, production-quality implementation. My work primarily deals with algorithms in the abstract, not so much with implementation-specific concerns. Hence, many of the algorithms have only been tested through abstract discrete-event simulations, which might not always capture the full picture. For instance, bandwidth restrictions in the network connecting PMs have only indirectly been accounted for. I suspect that there may be more to investigate in that area, both within cloud computing in general and in stateful serverless computing in particular.

On a more general note, as previously stated, some real-world problems unfortunately have multiple relevant bottleneck resources that cannot easily be transformed to single-objective problems via aggregate objective functions. It would be highly desirable to see an extension of the FIBP problem to multiple dimensions, together with with new algorithms which would retain as much as possible of the speed and accuracy offered by my algorithms for MIN-FIBP, but this lies more in the domain of operations research.

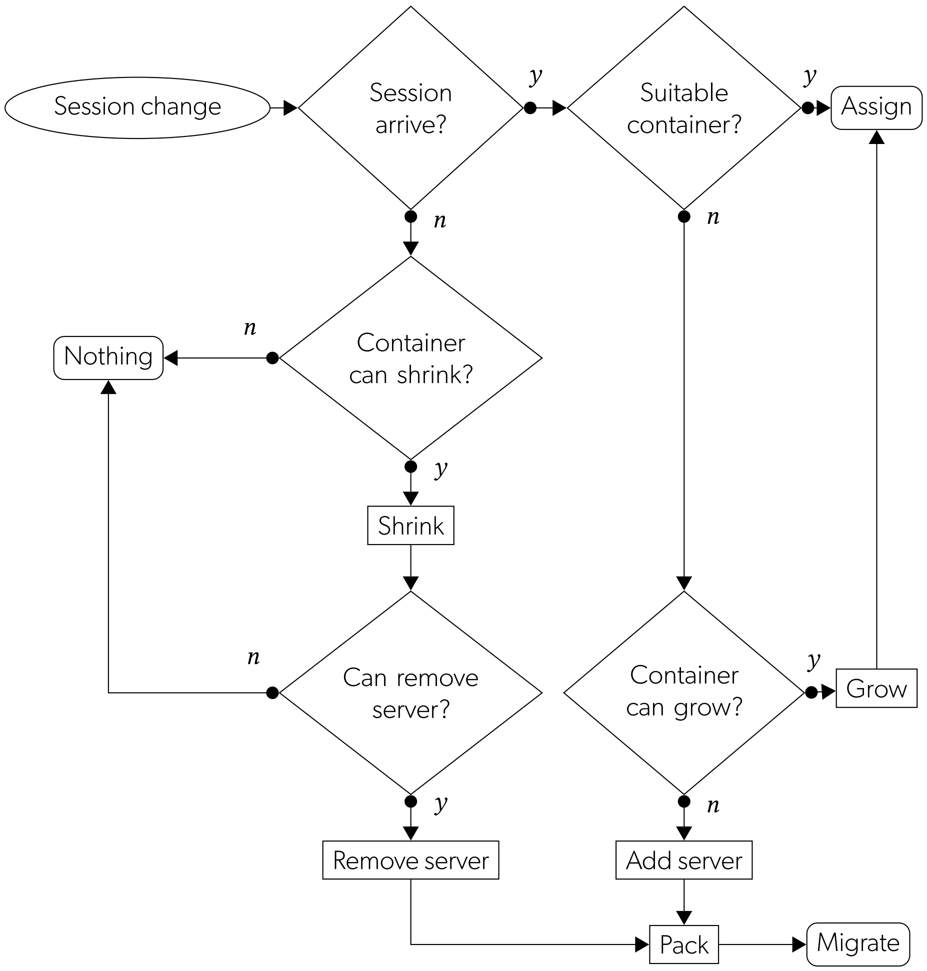


Figure 3.1: *The stateful serverless computing platform performs horizontal and vertical scaling combined with consolidation of servers and containers in an efficient manner, which avoids unnecessary effort.*

Glossary

ACID atomicity, consistency, isolation and durability

API application programming interface

BASE basically available soft state with eventual consistency

BP bin packing

CPU central processing unit

FaaS function as a service

FIBP fragmentable items bin packing

FIBP-EQ fragmentable items bin packing with equal capacities

GPGPU general-purpose computing on graphics processing units

GPU graphics processing unit

HTTP hypertext transfer protocol

IaaS infrastructure as a service

Min-FIBP minimum fragmentable items bin packing

Min-FIBP-EQ minimum fragmentable items bin packing with equal capacities

MIPS million instructions per second

MMKP multiple-choice multi-dimension knapsack problem

OLTP online transaction processing

PaaS platform as as service

PM physical machine

QoS quality of service

RAM random-access memory

SaaS software as a service

VM virtual machine

Bibliography

Ashraf, A., B. Byholm, J. Lehtinen, et al. (2012). "Feedback Control Algorithms to Deploy and Scale Multiple Web Applications per Virtual Machine." In *38th Euromicro Conference on Software Engineering and Advanced Applications*. Ed. by V. Cortellessa, H. Muccini, and O. Demirors. IEEE Computer Society, 431–438.

Ashraf, A., B. Byholm, and I. Porres (2012a). "A Session-Based Adaptive Admission Control Approach for Virtualized Application Servers." In *The 5th IEEE/ACM International Conference on Utility and Cloud Computing*. Ed. by C. Varela and M. Parashar. IEEE Computer Society, 65–72.

Ashraf, A., B. Byholm, and I. Porres (2012b). "CRAMP: Cost-Efficient Resource Allocation for Multiple Web Applications with Proactive Scaling." In *4th IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. Ed. by T. W. Wlodarczyk, C.-H. Hsu, and W.-c. Feng. IEEE Computer Society, 581–586.

Ashraf, A., B. Byholm, and I. Porres (2015). "A Multi-Objective ACS Algorithm to Optimize Cost, Performance, and Reliability in the Cloud." In *8th IEEE/ACM International Conference on Utility and Cloud Computing*. Ed. by O. Rana and M. Parashar. IEEE, 341–347.

Ashraf, A., B. Byholm, and I. Porres (2016). "Prediction-Based VM Provisioning and Admission Control for Multi-Tier Web Applications." *Journal of Cloud Computing*, 5.(1), pp. 1–21. ISSN: 2192-113X.

Ashraf, A., B. Byholm, and I. Porres (2018). "Distributed virtual machine consolidation: A systematic mapping study." *Computer Science Review*, 28C, pp. 118–130. ISSN: 1574-0137.

Baglama, J. and L. Reichel (2005). "Augmented Implicitly Restarted Lanczos Bidiagonalization Methods." *SIAM Journal on Scientific Computing*, 27.(1), pp. 19–42. DOI: 10.1137/04060593X.

Baldini, I. et al. (2017). "Serverless Computing: Current Trends and Open Problems." In *Research Advances in Cloud Computing*. Ed. by S. Chaudhary, G. Somani, and R. Buyya. Singapore: Springer Singapore, pp. 1–20. ISBN: 978-981-10-5026-8. DOI: 10.1007/978-981-10-5026-8_1.

Barroso, L. A. and U. Hölzle (2007). "The Case for Energy-Proportional Computing." *Computer*, 40.(12), pp. 33–37. DOI: 10.1109/MC.2007.443.

Beloglazov, A. and R. Buyya (Sept. 2012). “Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in Cloud data centers.” *Concurrency and Computation: Practice and Experience*, 24.(13), pp. 1397–1420. DOI: 10.1002/cpe.1867.

Byholm, B., F. Jokhio, et al. (2015). “Cost-Efficient, Utility-Based Caching of Expensive Computations in the Cloud.” In *23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. Ed. by M. Daneshtalab et al. Euromicro International Conference on Parallel, Distributed and Network-Based Computing. IEEE Computer Society Conference Publishing Services, 505–513.

Byholm, B. and I. Porres (2014). “Cost-Efficient, Reliable, Utility-Based Session Management in the Cloud.” In *14th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*. Ed. by P. Balaji et al. IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. IEEE Computer Society, 102–111.

Byholm, B. and I. Porres (2017a). *Dynamic Horizontal and Vertical Scaling of Multiple Cloud Services in Soft Real-Time*, tech. rep. 1182. TUCS.

Byholm, B. and I. Porres (2017b). “Optimized Deployment Plans for Platform as a Service Clouds.” In *Companion Proceedings of the 10th International Conference on Utility and Cloud Computing*. Ed. by G. Fox and Y. Chen. UCC ’17 Companion. ACM, 41–46.

Byholm, B. and I. Porres (Oct. 2018). “Fast algorithms for fragmentable items bin packing.” *Journal of Heuristics*, 24.(5), pp. 697–723. ISSN: 1572-9397. DOI: 10.1007/s10732-018-9375-z.

Calheiros, R. N. et al. (2011). “CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms.” *Software: Practice and Experience*, 41.(1), pp. 23–50. DOI: 10.1002/spe.995.

Deb, K. et al. (Apr. 2002). “A fast and elitist multiobjective genetic algorithm: NSGA-II.” *IEEE Transactions on Evolutionary Computation*, 6.(2), pp. 182–197. ISSN: 1089-778X. DOI: 10.1109/4235.996017.

Fan, X., W.-D. Weber, and L. A. Barroso (June 2007). “Power Provisioning for a Warehouse-Sized Computer.” *SIGARCH Comput. Archit. News*, 35.(2), pp. 13–23. ISSN: 0163-5964. DOI: 10.1145/1273440.1250665.

Fouladi, S. et al. (Mar. 2017). “Encoding, Fast and Slow: Low-Latency Video Processing Using Thousands of Tiny Threads.” In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. Boston, MA: USENIX Association, 363–376. ISBN: 978-1-931971-37-9.

Fox, A. et al. (2009). *Above the Clouds: A Berkeley View of Cloud Computing*, tech. rep.

Gale, W. A. and G. Sampson (1995). “Good-turing frequency estimation without tears.” *Journal of Quantitative Linguistics*, 2.(3), pp. 217–237. DOI: 10.1080/09296179508590051.

Goldberg, D. W. (Mar. 2009). “State considerations in distributed systems.” *XRDS*, 15.(3), pp. 7–11. ISSN: 1528-4972. DOI: 10.1145/1525902.1525905.

Hellerstein, J. M. et al. (2019). “Serverless Computing: One Step Forward, Two Steps Back.” In *Proceedings of the 9th Biennial Conference on Innovative Data Systems Research*. CIDR ’19. Asilomar, CA, USA: CIDR.

Jokhio, F. et al. (2013). “A Computation and Storage Trade-Off Strategy for Cost-Efficient Video Transcoding in the Cloud.” In *39th EUROMICRO Conference on Software Engineering and Advanced Applications*. Ed. by O. Demirors and O. Turetken. IEEE Computer Society, 365–372. DOI: 10.1109/SEAA.2013.17.

Kamra, A., V. Misra, and E. M. Nahum (2004). “Yaksha: A Self-Tuning Controller for Managing the Performance of 3-Tiered Web sites.” In *Twelfth IEEE International Workshop on Quality of Service*, 47–56. DOI: 10.1109/IWQOS.2004.1309356.

Klimovic, A. et al. (Oct. 2018). “Pocket: Elastic Ephemeral Storage for Serverless Analytics.” In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. Carlsbad, CA: USENIX Association, 427–444. ISBN: 978-1-939133-08-3.

Koller, R. and D. Williams (2017). “Will Serverless End the Dominance of Linux in the Cloud?” In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*. HotOS ’17. Whistler, BC, Canada: Association for Computing Machinery, 169–173. ISBN: 9781450350686. DOI: 10.1145/3102980.3103008.

LeCun, B. et al. (2015). “Bin packing with fragmentable items: Presentation and approximations.” *Theor. Comput. Sci.* 602, pp. 50–59. ISSN: 0304-3975. DOI: 10.1016/j.tcs.2015.08.005.

Ling, B. C., E. Kiciman, and A. Fox (2004). “Session State: Beyond Soft State.” In *Proceedings of the 1st Symposium on Networked Systems Design and Implementation (NSDI 2004)*. Ed. by R. T. Morris and S. Savage. USENIX, 295–308.

Liu, H. H. (2009). *Software Performance and Scalability: A Quantitative Approach*, Wiley Publishing. ISBN: 9780470462539.

Mell, P. and T. Grance (Sept. 2011). *SP 800-145. The NIST Definition of Cloud Computing*, tech. rep. Gaithersburg, MD, United States. DOI: 10.6028/NIST.SP.800-145.

Pérez, A. et al. (2018). “Serverless computing for container-based architectures.” *Future Generation Computer Systems*, 83, pp. 50–59. ISSN: 0167-739X. DOI: 10.1016/j.future.2018.01.022.

Quiroz Castellanos, M. et al. (2015). “A grouping genetic algorithm with controlled gene transmission for the bin packing problem.” *Comput. Oper. Res.* 55, pp. 52–64. ISSN: 0305-0548. DOI: 10.1016/j.cor.2014.10.010.

Shehabi, A. et al. (June 2016). *United States Data Center Energy Usage Report*, tech. rep. Berkeley, CA, United States. DOI: 10.2172/1372902.

Shen, H. and J. Z. Huang (2008). “Interday forecasting and intraday updating of call center arrivals.” *Manufacturing & Service Operations Management*, 10.(3), pp. 391–410. DOI: 10.1287/msom.1070.0179.

Toyoda, Y. (Aug. 1975). “A Simplified Algorithm for Obtaining Approximate Solutions to Zero-One Programming Problems.” *Manage. Sci.* 21.(12), pp. 1417–1427. ISSN: 0025-1909. DOI: 10.1287/mnsc.21.12.1417.

Van Eyk, E. et al. (2018). “A SPEC RG Cloud Group’s Vision on the Performance Challenges of FaaS Cloud Architectures.” In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*. ICPE ’18. Berlin, Germany: Association for Computing Machinery, 21–24. ISBN: 9781450356299. DOI: 10.1145/3185768.3186308.

Vetro, A., C. Christopoulos, and H. Sun (2003). “Video transcoding architectures and techniques: an overview.” *IEEE Signal Processing Magazine*, 20.(2), pp. 18–29. DOI: 10.1109/MSP.2003.1184336.

Vogels, W. (Jan. 2008). “Beyond Server Consolidation.” *Queue*, 6.(1), pp. 20–26. ISSN: 1542-7730. DOI: 10.1145/1348583.1348590.

Volgenant, A. (1996). “Linear and semi-assignment problems: A core oriented approach.” *Computers & Operations Research*, 23.(10), pp. 917–932. ISSN: 0305-0548. DOI: 10.1016/0305-0548(96)00010-X.

Von Neumann, J. and O. Morgenstern (1953). *Theory of Games and Economic Behavior*, Third. Princeton, NJ: Princeton University Press. 641 pp.

Benjamin Byholm

Optimizing Stateful Serverless Computing

Stateful serverless computing is a new paradigm within cloud computing. It successfully incorporates state management with serverless computing. Serverless computing is a form of cloud computing where the servers necessary for performing computation have been abstracted away, leaving the choice of where and how to perform a computation solely in the hands of the cloud provider. This abstraction simplifies the programming model for the cloud user, who can focus on business logic instead of scaffolding. It also offers the cloud provider greater freedom in how to manage the involved data centers, allowing for greater utilization of available resources.

In this thesis, I propose an autonomous platform for stateful serverless computing, provide a reference design and study the involved problems while providing their solutions. I focus on optimizing the entire system from the perspective of a cloud provider in terms of efficiency, cost and quality. The platform is able to autonomously adjust the supply of computing resources to meet fluctuations in demand without unnecessary waste. I show how to manage state in an efficient manner, which reduces latency while retaining flexibility in moving computations among servers. I further show how to manage a data cache in a cost-efficient manner, trading computation for storage. I present a new model for assigning computations to servers, allowing for higher utilization of available computing resources, thereby reducing the operational expenses of the cloud provider. I also show how to quickly solve this model, allowing for continuous redistribution of computations among servers to help maintain high resource utilization.

ISBN 978-952-12-4054-6