

# Performance Optimization for Cloud Computing Systems in the Microservice Era: State-of-the-Art and Research Opportunities

Rong Zeng, Xiaofeng Hou, Lu Zhang, Chao Li, Wenli Zheng, and Minyi Guo

Shanghai Jiao Tong University, Shanghai, China

© Higher Education Press and Springer-Verlag Berlin Heidelberg 2012

**Abstract** With the demand of agile development and management, cloud applications today are moving towards a more fine-grained microservice paradigm, where smaller and simpler functioning parts are combined for providing end-to-end services. In recent years, we have witnessed many research efforts that strive to optimize the performance of cloud computing system in this new era. This paper provides an overview of existing works on recent system performance optimization techniques and classify them based on their design focuses. We also identify open issues and challenges in this important research direction.

**Keywords** Microservice, Cloud Computing System, Performance Optimization, Challenges, Opportunities

## 1 Introduction

The cloud computing services today face stringent performance requirements, and meeting these requirements involves a variety of advanced techniques and significant research efforts. In an effort for agile cloud management and service development, many providers such as Netflix[1], Amazon[2], and Microsoft[3] are shifting to a new design paradigm — microservice. With microservice architecture, software applications are dissected to smaller, simpler functioning components that communicates with each other via network requests. Microservices can be developed via different programming models and languages, and deployed and managed independently without affecting the normal functions of each other[4][5].

Performance optimization of cloud computing system in the microservice era is challenging for several reasons. First, datacenters have grown in heterogeneity and disaggregation at the infrastructure layer due to the adoption of various special hardware [6, 7], accelerators[8] and new server architecture[9, 10]. Second, microservice-based applications have complex runtime dynamics. The service units <http://my.sjtu.edu.cn> are usually deployed in a distributed manner with lightweight virtualization[11]. Consequently, the complexity increases rapidly across all layers in the cloud, making it increasingly difficult for performance prediction, monitoring and resource provisioning decisions. Third, microservice applications have stricter performance requirements compared to the monolith. They show richer diversity in system requirements, which requires careful rethinking of current controlling techniques at both system and application level.

In this paper we compare recent techniques for microservice performance optimization in the cloud. We categorize performance optimization issues into three parts and investigate the latest work of each part to provide a holistic view of the problem. We mainly focus on techniques that address the resource management issue of cloud systems for high performance. We summarize the open issues and challenges posed by the complexity of microservices.

### 1.1 Related Work

While microservices have been widely adopted, only a few surveys have been conducted, and none of the prior surveys specifically focus on the performance optimization issue. Several literature reviews summarize recent work on microservice architectures. For example, Taibi et al. [12] presented the widely adopted architecture patterns and com-

pared the advantages and disadvantages of these patterns. Alshuqayran et al. [13] discussed the key concerns for designing architecture for microservice-based applications.

There are a few surveys on the security issues of microservice applications. Almeida et al. [14] investigated the security implications and solutions for cloud microservices. Yarygina et al. [15] collected the prior researches on security issues and presented a taxonomy in terms of the layered models and emergent practices.

One line of recent surveys has been focused on the study of differences between monolithic and microservice applications. Villamizar et al. [16] investigated monolithic and microservice for Web applications in cloud, with an emphasis on the architecture patterns. Vural et al. [17] looked at the trend of microservices and discussed the research gaps as well as the differences between microservice and SOA architecture. Gouigoux et al. [18] also compared the architecture and designs in monolithic and microservice Web services. As for production systems, Di Francesco et al. [19] summarized a few key industrial practices of microservice systems.

Service management approach is an important direction. Manvi et al. [20] presented the researches on the resource management approaches and optimizations at the infrastructure layer. Vaquero et al. [21] addressed the service orchestration challenges upon the advent of various new technologies, such as edge computing, serverless computing and disaggregated datacenters. Pahl et al. [22] presented the survey of microservices at the PaaS level. Our work distinguishes from these work in that we compare the latest work specifically addressing the performance optimization issues of clouding computing in the microservice era.

## 1.2 Contribution

It is important for researchers and developers to gain a deep understanding of potential performance bottlenecks and research challenges of microservice, as well as recent advancement on performance optimization approaches. In this paper we take the first step to extensively investigate various techniques that aim to improve microservice performance. Overall, this paper makes the following contributions:

- 1) We provide a thorough comparison of important terms and concepts that are closely related to cloud computing system in the microservice era.
- 2) We categorize the performance optimization techniques for microservice in the cloud, provide an analysis of the pros and cons of exiting designs, and present the research opportunities on these issues.

- 3) We summarize the open issues and challenges faced by cloud microservice management.

The remainder of this paper is organized as follows. Section 2 introduces the background of microservice architecture and related concepts, as well as the challenges and opportunities of performance optimization techniques. Sections 3, 4, and 5 provide an thorough review on performance monitoring, resource provisioning, and system tuning, respectively. Section 6 lists the open issues and challenges of microservice performance and adoption in cloud. Finally, Section 7 concludes the paper.

---

## 2 Background and Definition

This section provides an overview of microservice paradigm and its comparison with other related concepts.

### 2.1 Microservice: A Brief Overview

The term microservice was firstly introduced as "fine-grained" Service-Oriented Architecture (SOA) in 2012 [23], faced with the distributed nature of cloud system and applications [24]. In general, microservice is defined as an architectural style for software design, as opposed to the "monolithic" style [4]. In this paradigm, an application is composed of multi-tier, simpler-functioning services, each running in independent process and communicating through exposed ports to provide end-to-end services. To be more specific, the microservice architecture distinguishes itself from traditional cloud applications in the following aspects:

**Decoupled Functionality.** The applications are loosely decoupled into simpler functionalities. In the microservice paradigm each service is responsible for a pre-defined functionality, involving business logic, data retrieval, data storage, user request delivery, etc. The granularity and division of a microservice-based application is typically dependent on the process logic [4]. Here, the complete end-to-end functionality for user services is achieved by the collaboration of multiple microservice components through the network request of Remote Procedure Call (RPC) [25].

**Independent Development.** The microservice architecture demonstrates independence both in interior development approaches and communication methods. The developing of microservices usually involves the whole stack work and teams responsible for different services possess freedom in choosing languages and architecture patterns. It is important to leverage this diversity to enable agile goal-oriented software design.

**Table 1**  
Comparison of Different Computing Paradigms

Paradigms	Virtualization Level	Communication Mode	Language Diversity	Autonomous Management	Stateless
Microservice	C	D	Y	Y	Y
Service Oriented Architecture	B	C	N	N	N
Serverless Computing	C/V	D/C	Y	Y	Y
Event Driven Architecture	V	C	Y	N	Y
Model Driven Architecture	?	D/C	N	N	?

C - Container V - VM B-Bare-metal D - Decentralized C - Centralized Y - Yes N - No ? - Not specified

**Decentralized Management.** Guided by the principle of "share as little as possible" microservices are typically self-managed and distributed among virtual or physical resources [26]. This decentralization is especially beneficial for large systems since different parts of cloud could be scaled and updated as needed without the involvement of others [27]. Moreover, for decoupled data management, the system designers prefer individual database for each microservices.

## 2.2 Comparison of Related Concepts

The advent of microservice is a result of ubiquitous cloud computing plus the demand of decentralized application management and frequent service upgrade. Apart from the term "microservice", some more architectural patterns for software design and cloud management have been proposed.

**SOA.** Similar to microservices, Service-Oriented Architecture (SOA) features modularized design and message-based communication [28]. For SOA, services carry out small functions. An Enterprise Service Bus (ESB) is applied to connect services and consumers and shares data among all services, which makes SOA more like an integrated solution [29]. Microservice is different from SOA in that it is more decoupled: 1) In microservice architecture the services mostly communicate with each other through more lightweight network requests. Differently, in SOA the ESB is mainly responsible for intra-communication. 2) In microservice the data storage is decentralized as guided by the "share as little as possible" principle. However, in SOA the ESB is responsible for centralized data sharing. 3) The service components in Microservice system have individual access port for users and provide independent functions.

**Serverless Computing.** Serverless computing, is firstly proposed for peer-to-peer software platform, and has been attracting research focus since Amazon introduced the AWS Lambda [30]. Generally it has two variants in the scenario of cloud platform: "Function as a Service" (FaaS) rather than "Backend as a Service" (BaaS) [31]. In serverless com-

puting, the developers and cloud users do not have to consider the application deployment and environment configurations. The functions they run are registered by predefined performance requirements [32]. The cloud manager is responsible for deciding the proper infrastructure for each service/function to provide satisfying performance and users only pay for the number of executions of the functions and the compute power it consumes [31]. As the diversity of datacenter infrastructure increases, serverless computing can hide the details of underlying hardware and facilitate the cloud service deployment. According to the industrial survey [33], serverless computing is suitable for short-running and event-driven jobs and services [21, 34]. Microservice can be deployed in a serverless manner, and its difference from a serverless function, is that a microservice can consist of multiple functions [35]. While there are many explorations to use serverless deployment for microservice in a cost-effective and scalable way [36, 37], performance guarantee in this paradigm is a key research challenge [38].

**Event Driven Architecture (EDA).** The event-driven architecture is considered to be the foundation of the Microservice and FaaS [39]. In this model a service is evoked by the function call or the request from another service or end users. Services coordinate with each other by publishing and subscribing the events instead of binding the threads to requests throughout the whole execution process. Therefore, this paradigm enables loose coupling and highly distributed service rendering [40]. However, EDA brings challenges of the tracing [41] and performance guarantee [5, 42] in distributed systems.

**Model Driven Architecture (MDA).** MDA is frequently proposed in microservice development [43, 44]. It is firstly introduced as a model driven development approach [45], in which an abstraction of a system is used for designing a software application [46]. A model is a set of statements on system compositions and the relations with other systems [47]. In MDA, the microservice systems could be modeled by specific languages at different levels of abstractions. Mean-

while, the software implementations could be automatically generated by the description languages. Applying MDA in SOA has been addressed by prior researches [48, 49]. According to prior work [44], when it comes to using MDA for microservice development, there are some important design issues, such as modeling languages and tools.

### 2.3 Performance Optimization: An Overview

Performance optimization techniques for cloud computing systems in the microservice era are more challenging than conventional monolith-centric designs. Especially in large cloud systems, the performance states and request latency of fine-grained microservices are hard to track and monitor [5, 50]. Although cloud applications are decomposed into simpler service units that can be developed and deployed independently, it is unavoidable that the behavior of one microservice affects the others. The factors that have impacts on microservice performance can span across all layers, including software architecture and application design, service orchestration, resource capacity planning, runtime variation, and dynamic configuration of underlying hardware system. It is highly demanding to take an extensive investigation of performance optimization techniques from different aspects to provide a clear view of recent advances and research challenges. In this paper, we categorize the performance optimization problem into three aspects: performance evaluation and monitoring in the application layer, resource provisioning and management in the service layer, and system tuning and coordination in the infrastructure layer. We thoroughly review the proposed techniques and approaches in previous papers and present research opportunities and challenges in these three key areas, as shown in Figure 1.

**Performance Evaluation and Monitoring.** Microservice applications have finer-grained architecture and rich diversity in software architecture, interaction mode, and runtime environment. Understanding the impact of microservice on software/hardware has been the subject of many prior works. In this category, the research opportunities lie in understanding the implications across different layers in various runtime environments. It is also necessary to build efficient tools for performance monitoring and profiling. The research challenges in this part are summarized in Figure 1. First, due to the diversity and complexity of real-world microservice system, it is challenging to build benchmark systems that simulate the production environment, and uniform standards for performance evaluation. In addition, the service-level profiling dataset can be enormous; how to maintain and derive useful insights from the data is still an

open question.

**Resource Provisioning and Management.** For many years resource provisioning in datacenters remains to be a critical problem. To manage large-scale microservice system, cloud providers propose various tools and platforms to orchestrate applications and control resource allocation. These platforms generally require the users to configure resource requirements, and apply only basic strategies for service placement and resource adjustment. Moreover, in a highly dynamic runtime environment, the performance of some critical service can directly affect other services. Thus, making fast resource boosting decisions to accelerate services on the critical path is often critical. The research challenges in this part include understanding the diversified resource behaviors, analyzing service interactions, and building accurate models for performance prediction.

**System Tuning and Optimization.** Microservices have shorter latency compared to traditional cloud services. For system-level performance optimization, one of the research opportunities is to reduce the OS and network overhead to adapt to the microsecond-scale latency. Since many microservices are short-running tasks generally deployed in containers, some research efforts have been devoted on reducing the invocation cost. The research challenges in this part include designing new architectures, building appropriate virtual runtime environment, and identifying system-level performance bottlenecks.

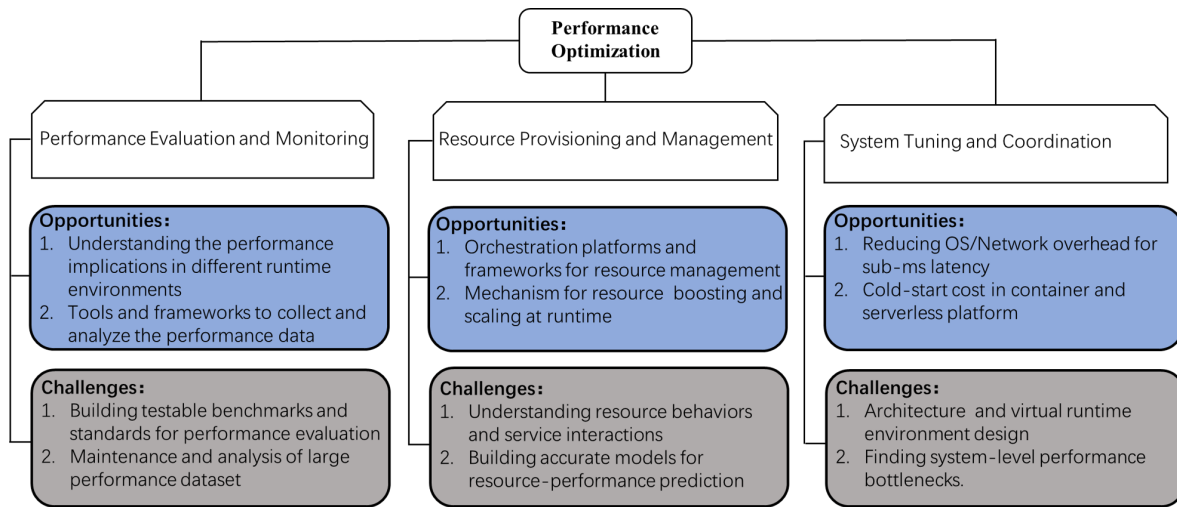
---

## 3 Performance Evaluation and Monitoring

Prior work on performance evaluation and monitoring of microservices have different design focuses that largely fall into three subcategories: benchmarking, analysis and characterization, monitoring and anomaly detection.

### 3.1 Benchmarking

Cloud service providers have developed simple open source benchmarks to facilitate microservice research, such as [51, 52]. Since previous researches use simple benchmarks, their results are not validated in production environment. Therefore, more sophisticated benchmarks are needed [53]. The benchmarks used for large system analysis and debugging, should have multiple service tiers, complex architecture, and meet basic business requirements. To facilitate the adoption and performance optimization of microservice in cloud, there is an urgent demand on the open source benchmarks that are easily accessible to research community [54]. There



**Fig. 1** Performance Optimization Techniques

are a few works that focus on the development and design of microservice system for various research purposes.

Aderaldo et al. [54] discussed the key requirements for cloud benchmark systems such as architecture patterns, version control and automation. While the scale of existing microservice system used by previous research is usually small, they are limited for performance debugging in real production environment. Towards more practical system design, Zhou et al. [55] developed a medium-size benchmark TrainTicket, which covers three interaction modes and four programming languages. Gan et al. [56] developed a benchmark suite, DeathStarBench. This suite includes five heterogeneous end-to-end microservice systems, consisting of tens of services each. The authors use the benchmark systems to explore the architectural implication of cloud microservices.

With the shift from the monolithic to microservices, modern On-Line Data Intensive (OLDI) applications are now facing a new sub-millisecond latency requirements [57]. Kratzke et al. [58] first proposed a benchmark to study the fundamental design for microservice. Sriraman et al. [59] presented a benchmark: Suite, to analyze how OS/network overheads affect the microservices which have sub-ms-scale latency. For performance testing and data collection in production deployment, Papapanagiotou et al. [60] proposed NDBench, a benchmarking tool that can automatically test the performance and system conditions at runtime.

### 3.2 Analysis and Characterization

The performance of a microservice application has different patterns compared to its monolithic counterparts. On the lower system layer, researches have been focused on

the characterization of microservices, including their fine-grained performance metrics, resource behaviors, and architectural implications. On the virtualization layer, there is a line of works that studies the performance impact of different virtualization technologies. On the application layer, different development options, such as architecture design, communication protocol, thread model, and their impacts on the application performance are addressed in some recent works.

**System and Architecture Implications.** Ueda et al. [61] compared the throughput of monolithic and microservice application, and also conducted experiments on the path length, cache misses and cycles per instruction of microservice system using different programming languages. Gan et al. [62] conducted extensive characterization on the Movie Streaming benchmark suite concerning the cycles distributions, instruction per second, and cache pressure. This analysis provides insight for cloud management as well as architectural design in the infrastructure layer. Sriraman et al. [63] characterized the top microservices used in Facebook clusters, and disclosed profound diversity among varied services in systems and architectural bottlenecks, such as hardware resource utilization, I/O interactions, and CPU stall behavior. Liu et al. [64] focused on the resource sensitivity of co-located microservices. The authors examined the QoS variation of widely used services under the allocation of different number of threads, cores, and LLC ways.

**Virtualization Method Comparison** The overhead brought by the virtualization layer prompted many researchers to look at efficient virtualization environment. Several popular virtualization platforms such as Unikernel, Docker, KVM are compared and evaluated concerning the

efficiency of concurrent provisioning multiple instances [65]. Saha et al. [66] evaluated the performance in a Docker deployment and bare-metal deployment. Jaramillo et al. [67] presented a case study for Docker applied in the microservice architecture. In a recent report [68] the architecture patterns leveraging the VM, container, and serverless platforms are compared and key limitations of these technologies are detailed. Kang et al. [69] compared the VM-based and container approach, and also the stateful and stateless microservice components. Lynn et al. presented the evolution of different virtualization technologies in [70] and conducted a multi-level analysis of cloud computing systems that leverage serverless and FaaS model. Esposito et al. [71] presented the designs of VM-based and container-based deployment and discussed concerns of different approaches. Villamizar et al. [72] compared the cost of different application patterns: Lambda functions, monolithic and microservices. The results show that, microservice pattern can help to reduce infrastructure cost, and the Lambda with the more granular provisioning, reduces the cost even more.

Some works focus on the system overhead and network performance impact brought by various virtualization and containerization technologies. Vastberg et al. [73] conducted a study on the overhead and performance impact incurred by Docker. Amaral et al. [74] analyzed the network performance concerning specifically to the traffic across hosts. The authors compared several deployment models for microservice: bare-metal, master-slave containerization, nested containerization and virtual machine. They also presented the network stack of each mechanism. Similarly, Kratzke et al. [75] proposed that the using containerized microservices can have performance impact due to the cost of hypervisor and software-defined networks(SDN). This paper presents experiment results, of how different factors impact the microservice networking performance, including the SDN, containerization, and encryption.

**Application Architecture Evaluation.** Microservice architecture shows flexibility in system design among different dimensions, including the communication methods and programming environment. Taibi et al. [12] detailed and compared several common communication patterns; further this work conducts an extensive study on the adoption of different patterns in cloud applications. Another survey on several communication protocols in the industry is conducted [76]. Saha et al. [66] evaluated the communication over virtual network in a containerized deployment for HPC workload. Sriraman et al. [59] examined the performance of different concurrency and communication model in OLDI

cloud applications. This paper also introduces a framework that can autonomously alter between synchronous and asynchronous models based on system load. Shadija et al. [77] conducted experiments to compare the performance of microservice application with different granularities and demonstrated that the deployment method and network latency have non-negligible impact on the balance between granularity and performance. Hassan et al. [26] identified the key factors considered by developers on the microservice granularity, and proposed some abstraction and evaluation approaches. Similarly, Lloyd et al. [78] investigated the factors that affect the microservice performance in a serverless deployment manner.

### 3.3 Monitoring and Anomaly Detection

Large cloud datacenters normally deploy performance monitoring and anomaly tracing. These topics have been widely studied in previous researches [79][80]. For microservice based application, there are some new challenges that need to be addressed. First, since microservice requests usually span across multiple service tiers and each service has different state and behavior under various configurations, it is necessary to devise new tools for tracing, profiling, and data collection. Second, microservice applications generally have complex architecture. We need mechanisms to derive insights of service behaviour and performance anomaly. To this end, some prior works have utilized sophisticated models for data trace analysis.

Gan et al. [5] presented an online monitoring and tracing system for more predictable performance and QoS violation. This work points out that the complicated dependencies among microservice components and the typical scale of today's large cloud systems often hinder performance prediction and quick adjustment; it instead leverages monitored large data and several machine learning models to predict the anomalies and the culprits. Nicol et al. [81] presented an online profiling tool; this proposed tool is compatible with different platforms, providing the collection of CPU and memory data in microservice-level and centralized data storage and analysis. Recently, Cinque et al [82] proposed a tracing tool, based on the analysis of REST message between microservices. Their method derives metrics from event logs, and therefore it is application transparent, without extra instrumentation and knowledge of application topology.

Sambasivan et al. [83] focused on the workflow-centric tracing and investigated the design space for tracing systems at different layers. In this work the authors pointed out that the tracing adoption should be reconsidered with the

**Table 2**  
Summary of Prior Works on Performance Evaluation and Monitoring

Categories	Subcategories	Prior Work
<b>Benchmarking</b>	\	[53], [56], [57], [60], [54], [55],[58]
<b>Analysis and Characterization</b>	System and Architecture Implications	[61], [62], [63], [64]
	Virtualization Method Evaluation	[65], [66], [67], [68], [69], [70], [71], [72], [73], [74], [75]
	Application Architecture Evaluation	[12], [76], [59], [77], [26]
<b>Monitoring and Anomaly Detection</b>	Tools & Frameworks	[83], [5], [81], [82]
	Analysis & Detection	[86], [87], [84], [85], [41]

infrastructure and different management tasks for best utilities. They delved the implementations of existing tracing system, and conducted an extensive systematic analysis on distributed services.

Performance anomaly detection and troubleshooting is another important research direction [84, 85]. There are studies that use graph model in microservice system to detect the root cause of performance degradation [86]. It presents an analysis framework which can perform troubleshooting based on a library of anomalous graphs. Lin et al. [87] leveraged the dependencies among services for detecting and pinpointing the culprit of performance anomaly. This work captures the network information and builds the connection among services to tracing the anomaly causes. Ravichandiran et al. [85] specifically focused on the resource consumption behaviors. The authors monitored the resource usage for services and built statistical model using the time series prediction methods to detect the anomalous behaviors and reduce the resource waste. Thalheim et al. [41] addressed specifically the challenges of the large volumes of data and high dimension of metrics, which hinders the cloud manager to derive insights for resource efficiency. In this paper, a framework is built to extract key metrics, reduce the amounts of data, and analyze the dependencies among distributed software components.

## 4 Resource Provisioning and Management

Appropriate resource provisioning and management are the key to ensure high performance. We group previous researches into three subcategories: 1) modeling and prediction, 2) placement and orchestration, 3) runtime adaptation.

### 4.1 Modeling and Prediction

One aspect of optimizing resource provisioning is to make capacity planning decisions based on reliable prediction models. This requires us to understand system performance under different resource configurations. Oftentimes, latency-sensitive services have wide workload runtime variation, with large fraction of low to moderate demand [88], and small fraction of peak loads [89]. Therefore, building accurate workload prediction model can be a challenging task.

Bao et al. [90] treated request execution time as the main performance metric. Specifically, this paper considers multiple factors that can contribute to response time, such as database accessing, networking failure, etc. The authors divided the execution into queuing, business processing and transaction processing. They also take the function sequence of executing a request into consideration, thereby deducting a precise model for request response time prediction. Jindal et al. [91] defined a performance metric MSC for detecting SLO violation. Based on this metric, they built the performance model by sandboxing a service and estimating MSC under different orchestration configuration.

One important line of work is to build non-linear model for performance prediction. For instances, Khazaei et al. [92] divided the problem into two parts: microservice platform analysis and macro-service infrastructure analysis. This work builds a Markov Chain model for the microservice platform, VM provisioning and PM provisioning, respectively. For example, for the microservice platform sub-model, it uses the number of requests, containers and VMs to denote the system status. Gribaudo et al. [93] modeled the microservice system as an oriented graph. They used Monte-carlo simulation to generate random application topologies. Then they derived the performance metrics for each generated scenario given VM allocation. Similarly Kannan et al [94] proposed to model the microservice-based multi-stage jobs as Directed Acyclic Graph (DAG), and used the DAGs

**Table 3**  
Summary of Prior Works on Resource Provisioning and Management

Categories	Subcategories	Prior Work
<b>Modeling &amp; Prediction</b>	Linear Model	[90], [91]
	Non-Linear Model	[95], [96], [92], [93], [94], [97], [85]
<b>Placement and Orchestration</b>	Application Layer	[99], [100], [96]
	Platform Layer	[69], [104], [105], [106]
<b>Runtime Adaptation</b>	Request Scheduling	[97], [90], [94]
	Resource Adjustment	[5], [64], [112], [42]
	Overload Control	[114], [115], [116]

to estimate the completion time of requests.

Queuing model has also been used for microservice response time prediction. For example, Correia et al. [95] modeled the application as a multi-server queue system, and the model can be used for resource planning with performance guarantee. Similarly, Yu et al. [96] used the M/M/c queue for the estimation of request processing time of a microservice. This work treats the request as a stream along microservice chain, and uses Poisson process to model the request arrival rate at each stage.

Note that request latency is not the only metric for performance modeling. Yan et al. [97] used the queue length of containers as the parameter of prediction model to derive resource utilization and the load of containers. Using this model the authors devised a decentralized strategy to schedule the messages. Similarly, Ravichandiran et al. [85] focused on the resource consumption behaviors. The authors monitored the resource usage for services and built statistical model using the time series prediction methods to detect anomalous behaviors and reduce resource waste. Hou et al. [98] built models for power consumptions of microservices in power-constrained data center.

#### 4.2 Placement and Orchestration

Microservices are generally deployed in light-weight virtual machines (VMs) or containers. The placement and orchestration of services have a non-trivial impact on the performance of the related application.

Some previous works mainly formulate microservice orchestration as an optimization problem, and use simulation platform to evaluate the proposed approach. Guerrero et al. [99] considered three key metrics in the orchestration for containers and services in the multi-cloud environment: operation cost, execution time, and repair time. This work derives an algorithm to decide the virtual machine allocation and container placement. Leng et al. [100] aimed to minimize the number of nodes used for microservice under QoS

requirements. The key method is to predict the resource usage of computing nodes based on the historical data. It then uses a pairwise ranking model to decide the service deployment. Yu et al [96] presented a scheme to jointly optimize the datacenter energy cost and service time. It features a three-stage method to search the decide the request routing and service instance placement. Klock et al. [101] presented a deployment scheme that utilizes the application features as well as dynamic performance statistics.

Several researches aim to address the design challenges on existing orchestration platforms or design new platforms. Platforms such as OpenStack [102], Kubernetes [103], are widely adopted for service orchestration in large environments. Kang et al. [69] mainly looked at the OpenStack platform and designed a deployment framework as an integrated component to enable service discovery. It redesigns the service state management and exposes container interfaces for accessing the device information. The qualification of this framework is conducted for both stateful and stateless containerized applications. Monteiro et al. [104] proposed an orchestration platform for complex event-driven microservices and presented a language to abstract the management option. Moreover, new platforms to ease cloud service orchestration are investigated as well [105, 106].

#### 4.3 Runtime Adaptation

Performance optimization at runtime for microservices requires a mixture of advanced resource adaptation techniques. On the application layer, request scheduling has been explored with different latency formulation models. Overload control is another import aspect for services to maintain normal response or scale before saturation, including mechanisms such as load shedding[107, 108] and resource boosting[109]. On the underlying infrastructure layer, different online schedulers have been proposed in recent years for monolithic cloud services[110, 111]. Recent works start to look at the various challenges posed by microservice appli-



cations, and use various models to make informed resource management decisions.

**Request Scheduling.** Yan et al. [97] used the queue length of containers as the parameter of prediction model to derive the resource utilization and load of containers. On top of this, the authors proposed a decentralized strategy to schedule the real-time messages. Bao et al. [90] modeled the user request as an execution of a directed acyclic graph. The authors formulated the performance prediction for each microservice based on its computing complexity. The whole application is modeled as a network of multiple functions, each of which could be executed inside a microservice instance. Thus, the request delay is formulated as the accumulation of execution time of its workflow. The authors then proposed several algorithm to solve the optimal request scheduling strategy to optimize end-to-end response time under pre-defined budget constraint. Kannan et al. [94] presented a holistic framework to execute multi-stage requests in microservice-based streamline application. This framework firstly predicts request completion time at each stage, and afterwards it uses the prediction to reorder requests based on the service-level agreement (SLAs). It also uses SLA-aware request batching to maximize throughput.

**Resource Adjustment.** Gan et al. [5] leveraged prediction mechanisms for resource adjustment. The authors utilized several machine learning models and large amount of historical data to predict QoS violation and pinpoint the culprit services. They proposed to re-allocate various hardware resources to reduce the load level. By analyzing large metric space and predicting the service state before the overload happens, this method effectively mitigates the QoS violation and precludes the cascading effects of service saturation. Liu [64] presented a runtime resource scheduler for microservice. The author uncovered the resource cliff phenomenon, which means that QoS is very sensitive with slight resource fluctuation. He conducted extensive analysis on the performance bottlenecks of widely used microservices, and used machine learning and reinforcement learning to make scheduling decisions. Alipour et al. [112] focused on the resource pattern for microservices and leveraged the machine learning for resource demand predication to mitigate low utilization. Cui et al. [42] focused on the asynchronous event-based service model. They presented a solution to pinpoint the latency bottleneck using event dependency graph and used an energy-efficient boosting technique to optimize the tail latency. The techniques proposed by Chang et al. [113] aim to facilitate the automatic resource adjustment. The authors conducted empirical measurements

on microservice performance by purposely throttling hardware resources. They used resource-performance correlation to pinpoint the resources that may benefit the service most.

**Overload Control.** Zhou et al. [114] proposed a control scheme to address the key challenges in large scale microservice backend system. The authors pointed out that monitoring the states of each microservice in a highly dynamic distributed environment is intractable, and the service dependencies make it hard to execute the control mechanism independently. The presented scheme tackles these challenges using several techniques. First, it maintains an adaptive algorithm and threshold at each microservice. Second, it executes the load shedding to the upstream microservices. This scheme is service agnostic and requires minimum coordination between services. Suresh et al. [115] also explored practical solutions for overload control problem in the SOA and microservice application where complex dependencies exist. Moreover, this work addresses the multi-tenancy resource sharing in cloud applications. It models each request as a DAG, and applies two techniques: workflow level rate limiting and request-level scheduling. The proposed scheme tracks service utilization and request execution time. It schedules request based on its latency slack to achieves a desired trade-off between utilization and load shedding. Xu et al. [116] used a brownout approach for large-scale production environment. When the system is overloaded, this approach deactivates part of the microservice components and the remaining part is still functioning.

---

## 5 System Tuning and Coordination

The disaggregation at the application layer of microservice results in diversified requirements for system and architecture [63]. Meanwhile, microservice applications have much shorter request service time, typically in the microsecond-scale [117]. These factors call for re-thinking of system-level designs that can enhance the efficiency of execution and communication, and enable more agile system coordination across the computing stack.

Tail latency is the primal design target of cloud datacenters in many prior works [59, 117, 118]. The short latency of microservice requests poses new challenges on existing power management frameworks. For millisecond-scale services, conventional designs often take advantage of the latency slack and the arrival intervals between requests to control processor cores to run in different power modes (frequency/voltage levels) [119, 120, 121]. However,

**Table 4**  
Summary of Prior Works on System Tuning and Coordination

Optimization Target	Ref.	Methodology	Main Results
Tail Latency	[117]	Coordinate CPU power state and frequency scaling, to meet the $\mu s$ scale latency just in time	Achieve stable tail latency and save CPU energy consumption for microservice
	[118]	Extend the simultaneously multi-threaded core to schedule short requests	The proposed mechanism reduces tail latency under moderate and high load
	[59]	Automatically tuning the threading model for mid-tier microservice based on offline profiling	The proposed $\mu$ Tune shows $1.9 \times$ tail latency speedup over static threading choices
Invocation Latency	[122]	Using language-level isolation and fast preemption to reduce the invocation latency	Language-level isolation and intra-process preemption can be used for $\mu s$ scale latency
	[123]	Using lightweight container isolation and package-aware caching to reduce the invocation latency	The proposed SOCK achieves significant speedup over Docker
Communication Overhead	[124]	Using application sandboxing and a hierarchical message bus to speed up service communication	The proposed SAND achieves a 43% speedup over OpenWhisk
	[125]	Using a userspace networking stack that constructs fast channels between partner microservices	The proposed Docknet achieves high throughput and low latency

these approaches may have very limited effectiveness, since the microsecond-scale request interval results in fragmented idle periods and insufficient time for power mode transition. Chou et al. [117] discussed the challenges brought by the microsecond-scale request arrival time and processing time. This paper proposes to coordinate frequency scaling, request packing, and power states techniques, to prolong the idle periods and guarantee performance. Differently, Mirhosseini et al. [118] pointed out that the queuing process is a main part of tail latency. The short tasks of microservices cause high contention and synchronization costs in a scale-up system. This paper proposes Express-Lane SMT (ESMT) to provide shorter execution queue for short tasks. It does this by extending the simultaneously multithreaded (SMT) core to distribute tasks based on their length. The proposed ESMT enables microservice tasks run in a scale-out architecture, and minimizes queuing delay and tail latency.

Since microservices typically have sub-microsecond latency requirement, the OS-level overhead—such as thread invoking and context switch, become non-trivial factors to tail latency. Sriraman et al. [59] explored the tail latency improvement in distributed microservices, specifically, On-Line Data Intensive (OLDI) applications. This work points out that modern microservices usually have sub-microsecond latency SLOs since a user request typically involves many services. Therefore the threading and concur-

rency model, which could have salient impacts on the response latency, are more critical when it comes to microservices. This paper does an thorough analysis on different concurrency designs. It proposes a framework to automatically switch the thread according to load variation, resulting in improved tail latency for multi-tier microservices.

Several papers propose techniques to reduce invocation latency, which refers to the cost to warm up a process or container. Boucher et al. [122] observed two bottlenecks that stall the microservices from achieving the microsecond-scale latency: process-based isolation and millisecond preemption. The authors proposed a language-based isolation to secure the microservices executed in shared process, improving the latency significantly; additionally they proposed the fast preemption for intra-process microservices to improve the latency and throughput further. Oakes et al. [123] addressed the long start-up latency of containers. In this paper, several key observations are made concerning the factors contributing to the long start-up. A container system is implemented to address the latency issues by: 1) using namespace and cgroup isolation to mitigate the performance bottleneck; 2) using the full language repositories on worker machines to avoid Python initialization cost; 3) designing a multi-tier caching system. The presented system has significant improvement in the cold-start latency and throughput.

Inter-service communication overhead has drawn much

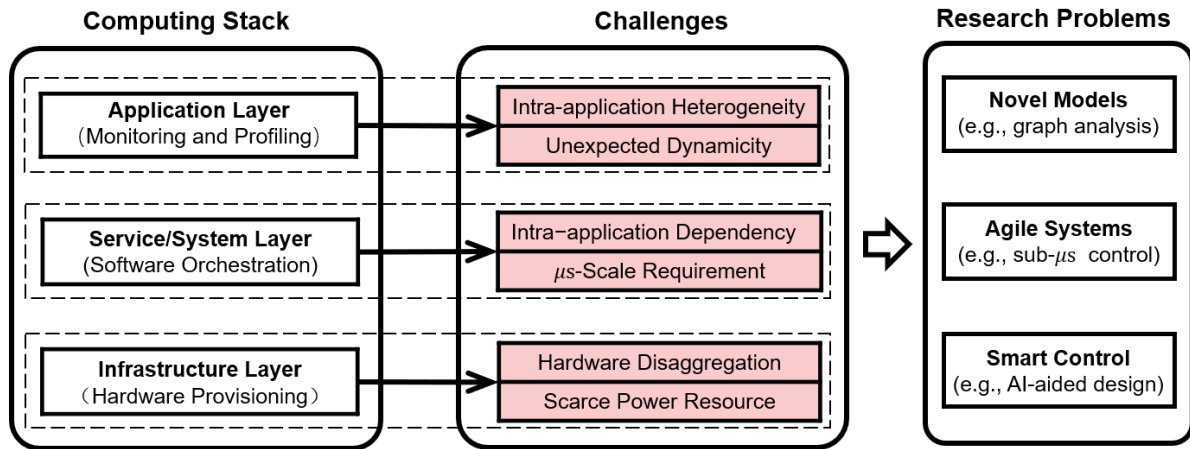


Fig. 2 Open Issues in The Microservice Era

attention as distributed microservices often run across multiple containers or even involves multi-server coordination [4]. Akkus et al. [124] pointed out two factors that could affect the efficiency of container orchestration: the deployment of functions in container instance and the interaction among functions in different instances. This work proposes a two-level isolation scheme to facilitate the functions of the same application. It also uses a hierarchical message queue to reduce the inter-node communication cost. Luo et al. [125] showed that the inter-service communication incurs additional network pressure and that containerization has brought inefficiency. They proposed a mechanism, DockNet, aiming to improve the network performance. DockNet is a userspace networking stack that uses rate limiters for performance isolation between different containers. It constructs fast channels between partner microservice to optimize the inter-service communication.

## 6 Research Opportunities

Efficiently supporting microservice-based applications in the cloud is a challenging task. In this section, we summarize key open issues brought by microservice. In Figure 2, we present the main challenges lied in different cloud computing layers and discuss the potential solutions.

### 6.1 New Models and Abstractions

While the microservice architecture has brought great flexibility in application development and deployment, it also vitiates the effectiveness of existing system optimization approaches used for monolithic applications. For example, applications face service components with various depen-

dencies, each of which can be written in different programming frameworks and deployed in different container technologies. Since future applications exhibit drastically different workload patterns, techniques for microservice-level modelling and workload prediction are required. Moreover, as we introduce more specified and disaggregated hardware/accelerators, the heterogeneity of datacenter is intensified. It is important to build new models to analyze the interplay between applications and underlying resources

### 6.2 System Orchestration on $\mu$ s-Scale

Microservice has brought the computer system into the "era of the killer microseconds". It is easy for programmers to mitigate event latency in the nanosecond and millisecond time scales (such as DRAM accesses at tens or hundreds of nanoseconds and disk I/O at a few milliseconds). However, little work has been done in terms of supporting for microsecond  $\mu$ s-scale events. Previous researches have shown that the sub-millisecond system overhead (like thread switching) has non-trivial impact on microservice latency. Therefore, it is essential to design systems that can better support concurrency, frequency scaling and I/O mechanism. In particular, conventional power management in large-scale systems incurs long latency. It is also of great interest to quantify the latency of power management, identify the culprit of performance degradation and eliminate latency.

### 6.3 Autonomous Resource Management

Designing system mechanisms that can adaptively and autonomously manage the underlying computing resource to meet the needs of microservices are becoming increasingly important. Oftentimes, the operators and designers of a

cloud datacenter need to search for the desired machine configuration and resource allocation. The performance requirements differ across various services and resources, and the number of configurations can be huge. Without appropriate design, searching for scheduling decisions under certain performance constraints can be time-consuming. It would be beneficial to leverage the state-of-the-art machine learning techniques to guide the management of the underlying computing resources. In addition, the computing environments can quickly change as applications scale out or migrate. To continuously make smart scheduling decisions in a complex operating environment, novel autonomous control techniques need to be developed. In other words, the system designed for microservice-based applications needs to have better self-managing capability.

#### 6.4 Cross-Layer Design

Eventually, we expect that a microservice-oriented, cross-layer coordination scheme is needed for datacenter-scale computing in the near future. Today, various resource management techniques are applied separately at different layers of the cloud computing stack. While emerging hardware components are deployed at the lower infrastructure level, the upper level software systems are not able to leverage this flexibility due to a lack of coordination. As we transition from conventional monolith to microservices, it is crucial to develop a deep understanding of resource sharing behavior at different computing layers (hardware, architecture, operating system, middleware, etc.). In particular, designing agile and adaptive coordination mechanisms for fine-grained, short-lived services is of great importance.

## 7 Conclusion

The burgeoning of various cloud applications continues to push the advancement of datacenter-scale computing. Recently with the maturity of virtualization technologies and realization of microsecond-scale network latency, microservice architecture has shown great promise in cloud application development. Microservice not only brings new opportunities, but also renders it more challenging to meet the performance requirements. In this article, we draw a detailed review of the latest work on performance optimization for microservice. We investigate key design considerations of cloud microservices, summarize representative design approaches, and identify the open issues that need to be addressed. We hope that this survey can shed some light on

microservice research today and tomorrow.

**Acknowledgements** We thank all the reviewers for their valuable comments and feedbacks. This work is sponsored by the National Natural Science Foundation of China (No. 61972247). Corresponding author is Chao Li from Shanghai Jiao Tong University.

---

## References

- [1] Adopting microservices at netflix: Lessons for team and process design. <https://www.nginx.com/learn/microservices/>.
- [2] Microservices architectures on amazon web services. <https://docs.aws.amazon.com/whitepapers/latest/microservices-on-aws/simple-microservices-architecture-on-aws.html>.
- [3] Microservices in Azure. <https://azure.microsoft.com/en-us/solutions/microservice-applications/>.
- [4] Microservices. <https://martinfowler.com/articles/microservices.html>.
- [5] Yu Gan, Yanqi Zhang, Kelvin Hu, Yuan He, Meghna Pancholi, Dailun Cheng, and Christina Delimitrou. Seer: Leveraging Big Data to Navigate the Complexity of Performance Debugging in Cloud Microservices. In *Proceedings of the Twenty Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, April 2019.
- [6] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, et al. Dadiannao: A machine-learning supercomputer. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 609–622. IEEE Computer Society, 2014.
- [7] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pages 1–12. IEEE, 2017.

- [8] Eric Chung, Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Adrian Caulfield, Todd Mas-sengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, et al. Serving dnns in real time at datacenter scale with project brainwave. *IEEE Micro*, 38(2):8–20, 2018.
- [9] Vlad Nitu, Boris Teabe, Alain Tchana, Canturk Isci, and Daniel Hagimont. Welcome to zombieland. In *Proceedings of the Thirteenth EuroSys Conference on - EuroSys '18*, pages 1–12, New York, New York, USA, 2018. ACM Press.
- [10] Kevin Lim, Jichuan Chang, Trevor Mudge, Parthasarathy Ranganathan, Steven K Reinhardt, and Thomas F Wenisch. Disaggregated memory for expansion and sharing in blade servers. *ACM SIGARCH Computer Architecture News*, 37(3):267, 2009.
- [11] Serverless deployment. <https://microservices.io/patterns/deployment/serverless-deployment.html>.
- [12] Davide Taibi, Valentina Lenarduzzi, and Claus Pahl. Architectural patterns for microservices: A systematic mapping study. In *CLOSER*, pages 221–232, 2018.
- [13] Nuha Alshuqayran, Nour Ali, and Roger Evans. A systematic mapping study in microservice architecture. In *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*, pages 44–51. IEEE, 2016.
- [14] Washington Henrique Carvalho Almeida, Luciano de Aguiar Monteiro, Raphael Rodrigues Hazin, Anderson Cavalcanti de Lima, and Felipe Silva Ferraz. Survey on microservice architecture-security, privacy and standardization on cloud computing environment. *ICSEA 2017*, page 210, 2017.
- [15] Tetiana Yarygina and Anya Helene Bagge. Overcoming security challenges in microservice architectures. In *2018 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, pages 11–20. IEEE, 2018.
- [16] Mario Villamizar, Oscar Garces, Harold Castro, Mauricio Verano, Lorena Salamanca, Rubby Casas-las, and Santiago Gil. Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. In *2015 10th Computing Colombian Conference (10CCC)*, pages 583–590. IEEE, sep 2015.
- [17] Hulya Vural, Murat Koyuncu, and Sinem Guney. A systematic literature review on microservices. In *International Conference on Computational Science and Its Applications*, pages 203–217. Springer, 2017.
- [18] Jean-Philippe Gouigoux and Dalila Tamzalit. From monolith to microservices: Lessons learned on an industrial migration to a web oriented architecture. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, pages 62–65. IEEE, 2017.
- [19] Paolo Di Francesco, Patricia Lago, and Ivano Malavolta. Migrating Towards Microservice Architectures: An Industrial Survey. In *2018 IEEE International Conference on Software Architecture (ICSA)*, pages 29–2909. IEEE, apr 2018.
- [20] Sunilkumar S Manvi and Krishna Shyam. Resource management for Infrastructure as a Service (IaaS) in cloud computing: A survey. 2014.
- [21] Jorge Bernal-Bernabe, Satish N. Srirama, Luis M. Vaquero, Mohamed Faten Zhani, Yehia Elkhatib, and Felix Cuadrado. Research challenges in nextgen service orchestration. *Future Generation Computer Systems*, 90:20–38, 2018.
- [22] Claus Pahl and Pooyan Jamshidi. Microservices: A systematic mapping study. In *CLOSER (1)*, pages 137–146, 2016.
- [23] James Lewis. Main sponsor Micro Services-Java the Unix way. Technical report.
- [24] Learn from SOA: 5 lessons for the microservices era. <https://www.infoworld.com/article/3080611/>.
- [25] Apache thrift. <https://thrift.apache.org/>.
- [26] Sara Hassan and Rami Bahsoon. Microservices and Their Design Trade-Offs: A Self-Adaptive Roadmap. In *2016 IEEE International Conference on Services Computing (SCC)*, pages 813–818. IEEE, jun 2016.
- [27] Giovanni Toffetti, Sandro Brunner, Martin Blöchl-inger, Florian Dudouet, and Andrew Edmonds. An

- architecture for self-managing microservices. In *Proceedings of the 1st International Workshop on Automated Incident Management in Cloud*, pages 19–24. ACM, 2015.
- [28] Bob Familiar. Microservice Architecture. In *Microservices, IoT, and Azure*, pages 21–31. 2015.
- [29] Pooyan Jamshidi, Claus Pahl, Nabor C Mendonça, James Lewis, and Stefan Tilkov. Microservices: The journey so far and challenges ahead. *IEEE Software*, 35(3):24–35, 2018.
- [30] NEW LAUNCH: Getting Started with AWS Lambda - YouTube.
- [31] Microservice and Serverless Computing. <https://www.endava.com/en/blog/Engineering/2019/Microservices-and-Serverless-Computing>.
- [32] Ioana Baldini, Paul Castro, Kerry Chang, Perry Cheng, Stephen Fink, Vatche Ishakian, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, Aleksander Slominski, et al. Serverless computing: Current trends and open problems. In *Research Advances in Cloud Computing*, pages 1–20. Springer, 2017.
- [33] Geoffrey C Fox, Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. Status of serverless computing and function-as-a-service (faas) in industry and research. *arXiv preprint arXiv:1708.08028*, 2017.
- [34] Paul Castro, Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. Serverless Programming (Function as a Service). In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 2658–2659. IEEE, jun 2017.
- [35] What Is a Serverless Microservice? <https://www.cloudflare.com/learning/serverless/glossary/serverless-microservice/>.
- [36] Mengting Yan, Paul Castro, Perry Cheng, and Vatche Ishakian. Building a chatbot with serverless computing. pages 1–4, 12 2016.
- [37] Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. Serving deep learning models in a serverless platform. *CoRR*, abs/1710.08460, 2017.
- [38] Paul Castro, Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. The rise of serverless computing. *Commun. ACM*, 62(12):44–54, November 2019.
- [39] Geoffrey C. Fox, Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. Status of serverless computing and function-as-a-service(faas) in industry and research. *CoRR*, abs/1708.08028, 2017.
- [40] Brenda M Michelson. Event-driven architecture overview. *Patricia Seybold Group*, 2(12):10–1571, 2006.
- [41] Jörg Thalheim, Antonio Rodrigues, Istemi Ekin Akkus, Pramod Bhatotia, Ruichuan Chen, Bimal Viswanath, Lei Jiao, and Christof Fetzer. Sieve: actionable insights from monitored metrics in distributed systems. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*, pages 14–27. ACM, 2017.
- [42] Wenzhi Cui, Daniel Richins, Yuhao Zhu, and Vijay Janapa Reddi. Tail latency in node.js: Energy efficient turbo boosting for long latency requests in event-driven web services. In *Proceedings of the 15th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE 2019, pages 152–164, New York, NY, USA, 2019. ACM.
- [43] Branko Terzić, Vladimir Dimitrieski, Slavica Korđić (Aleksić, and Ivan Luković. A model-driven approach to microservice software architecture establishment. pages 73–80, 09 2018.
- [44] Florian Rademacher, Sabine Sachweh, and Albert Zündorf. Differences between model-driven development of service-oriented and microservice architecture. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, pages 38–45. IEEE, 2017.
- [45] Richard Soley et al. Model driven architecture. *OMG white paper*, 308(308):5, 2000.
- [46] Alberto Rodrigues da Silva. Model-driven engineering: A survey supported by the unified conceptual model. *Computer Languages, Systems Structures*, 43:139 – 155, 2015.
- [47] Edwin Seidewitz. What models mean. *IEEE software*, 20(5):26–32, 2003.

- [48] Ken Laskey, Jeff A Estefan, Francis G McCabe, and D Thornton. Reference architecture foundation for service oriented architecture version 1.0. *Oasis, Committee Draft*, 2:26, 2009.
- [49] David Ameller, Xavier Burgués, Oriol Collell, Dolors Costal, Xavier Franch, and Mike P Papazoglou. Development of service-oriented architectures using model-driven development: A mapping study. *Information and Software Technology*, 62:42–66, 2015.
- [50] Rajiv Ranjan, Chang Liu, Lydia Chen, Maria Fazio, Massimo Villari, and Antonio Celesti. Open Issues in Scheduling Microservices in the Cloud. *IEEE Cloud Computing*, 3(5):81–88, 2016.
- [51] Socks shop – a microservices demo application. <https://microservices-demo.github.io/>.
- [52] Sample musicstore application. <https://github.com/aspnet/MusicStore>.
- [53] Nuha Alshuqayran, Nour Ali, and Roger Evans. A systematic mapping study in microservice architecture. In *Proceedings - 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications, SOCA 2016*, pages 44–51. IEEE, nov 2016.
- [54] Carlos M. Aderaldo, Nabor C. Mendonça, Claus Pahl, and Pooyan Jamshidi. Benchmark Requirements for Microservices Architecture Research. *Proceedings - 2017 IEEE/ACM 1st International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering, ECASE 2017*, pages 8–13, 2017.
- [55] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chao Ji, Wenhai Li, and Dan Ding. Fault Analysis and Debugging of Microservice Systems: Industrial Survey, Benchmark System, and Empirical Study. *IEEE Transactions on Software Engineering*, pages 1–1, 2018.
- [56] Yu Gan, Yanqi Zhang, Dailun Cheng, Ankitha Shetty, Priyal Rathi, Nayan Katarki, Ariana Bruno, Justin Hu, Brian Ritchken, Brendon Jackson, et al. An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 3–18. ACM, 2019.
- [57] A. Sriraman and T. F. Wenisch.  $\mu$ suite: A benchmark suite for microservices. In *2018 IEEE International Symposium on Workload Characterization (IISWC)*, pages 1–12, Sep. 2018.
- [58] Nane Kratzke and Peter-Christian Quint. Ppbench. In *Proceedings of the 6th International Conference on Cloud Computing and Services Science - Volume 1 and 2, CLOSER 2016*, page 223–231, Setubal, PRT, 2016. SCITEPRESS - Science and Technology Publications, Lda.
- [59] Akshitha Sriraman and Thomas F. Wenisch. tune: Auto-tuned threading for OLDI microservices. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 177–194, Carlsbad, CA, 2018. USENIX Association.
- [60] Ioannis Papapanagiotou and Vinay Chella. Ndbench: Benchmarking microservices at scale. *CoRR*, abs/1807.10792, 2018.
- [61] T. Ueda, T. Nakaike, and M. Ohara. Workload characterization for microservices. In *2016 IEEE International Symposium on Workload Characterization (IISWC)*, pages 1–10, Sep. 2016.
- [62] Yu Gan and Christina Delimitrou. The Architectural Implications of Cloud Microservices. *IEEE Computer Architecture Letters*, 17(2):155–158, 2018.
- [63] Akshitha Sriraman, Abhishek Dhanotia, and Thomas F Wenisch. Softsku: optimizing server architectures for microservice diversity@ scale. In *Proceedings of the 46th International Symposium on Computer Architecture*, pages 513–526. ACM, 2019.
- [64] Lei Liu. Qos-aware machine learning-based multiple resources scheduling for microservices in cloud environment. *arXiv preprint arXiv:1911.13208*, 2019.
- [65] Bruno Xavier, Tiago Ferreto, and Luis Jersak. Time Provisioning Evaluation of KVM, Docker and Unikernels in a Cloud Platform. In *Proceedings - 2016 16th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, CCGrid 2016*, pages 277–280, 2016.

- [66] Pankaj Saha, Angel Beltré, Piotr Uminski, and Madhusudhan Govindaraju. Evaluation of Docker Containers for Scientific Workloads in the Cloud. pages 1–8, 2018.
- [67] David Jaramillo, Duy V Nguyen, and Robert Smart. Leveraging microservices architecture by using docker technology. In *SoutheastCon 2016*, pages 1–5. IEEE, 2016.
- [68] Evolution of Server Computing: VMs to Containers to Serverless - Which to Use When?
- [69] Hui Kang, Michael Le, and Shu Tao. Container and microservice driven design for cloud infrastructure devops. In *2016 IEEE International Conference on Cloud Engineering (IC2E)*, pages 202–211. IEEE, 2016.
- [70] Theo Lynn, Pierangelo Rosati, Arnaud Lejeune, and Vincent Emeakaroha. A preliminary review of enterprise serverless cloud computing (function-as-a-service) platforms. In *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 162–169. IEEE, 2017.
- [71] Christian Esposito, Aniello Castiglione, and Kim-Kwang Raymond Choo. Challenges in delivering software in the cloud as microservices. *IEEE Cloud Computing*, 3(5):10–14, 2016.
- [72] Mario Villamizar, Oscar Garces, Lina Ochoa, Harold Castro, Lorena Salamanca, Mauricio Verano, Rubby Casallas, Santiago Gil, Carlos Valencia, Angee Zambrano, et al. Infrastructure cost comparison of running web applications in the cloud using aws lambda and monolithic and microservice architectures. In *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 179–182. IEEE, 2016.
- [73] Anders Västberg. Container overhead in microservice systems Container overhead in microservice systems. 2018.
- [74] Marcelo Amaral, Jordà Polo, David Carrera, Iqbal Mohamed, Merve Unuvar, and Malgorzata Steinder. Performance evaluation of microservices architectures using containers. In *Proceedings - 2015 IEEE 14th International Symposium on Network Computing and Applications, NCA 2015*, pages 27–34. IEEE, sep 2016.
- [75] Nane Kratzke. About microservices, containers and their underestimated impact on network performance. *CoRR*, abs/1710.04049, 2017.
- [76] Felipe Osses, Gastón Márquez, and Hernán Astudillo. Exploration of academic and industrial evidence about architectural tactics and patterns in microservices. pages 256–257, 2018.
- [77] Dharmendra Shadija, Mo Rezai, and Richard Hill. Microservices: Granularity vs. Performance. Technical report.
- [78] Wes Lloyd, Shruti Ramesh, Swetha Chinthlapati, Lan Ly, and Shrideep Pallickara. Serverless computing: An investigation of factors influencing microservice performance. In *2018 IEEE International Conference on Cloud Engineering (IC2E)*, pages 159–169. IEEE, 2018.
- [79] Hyunwook Baek, Abhinav Srivastava, and Jacobus Van der Merwe. Cloudsight: A tenant-oriented transparency framework for cross-layer cloud troubleshooting. In *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 268–273. IEEE Press, 2017.
- [80] Guilherme Da Cunha Rodrigues, Rodrigo N Calheiros, Vinicius Tavares Guimaraes, Glederson Lessa dos Santos, Marcio Barbosa De Carvalho, Lisandro Zambenedetti Granville, Liane Margarida Rockenbach Tarouco, and Rajkumar Buyya. Monitoring of cloud computing environments: concepts, solutions, trends, and future directions. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, pages 378–383. ACM, 2016.
- [81] John Nicol, Chen Li, Peinan Chen, Tao Feng, and Haricharan Ramachandra. ODP: An Infrastructure for On-Demand Service Profiling. In *ICPE*, 2018.
- [82] Marcello Cinque, Raffaele Della Corte, and Antonio Pecchia. Microservices monitoring with event logs and black box execution tracing. *IEEE Transactions on Services Computing*, 2019.
- [83] Raja R. Sambasivan, Ilari Shafer, Jonathan Mace, Benjamin H. Sigelman, Rodrigo Fonseca, and Gregory R. Ganger. Principled workflow-centric tracing of distributed systems. In *Proceedings of the Seventh ACM Symposium on Cloud Computing, SoCC '16*, pages 401–414, New York, NY, USA, 2016. ACM.



- [84] Thomas F Düllmann. Performance anomaly detection in microservice architectures under continuous change. Master's thesis, 2017.
- [85] Rajsimman Ravichandiran, Hadi Bannazadeh, and Alberto Leon-Garcia. Anomaly Detection using Resource Behaviour Analysis for Autoscaling systems. *2018 4th IEEE Conference on Network Softwarization and Workshops, NetSoft 2018, (NetSoft):267–271*, 2018.
- [86] Álvaro Brandón, Marc Solé, Alberto Huélamo, David Solans, María S Pérez, and Victor Muntés-Mulero. Graph-based root cause analysis for service-oriented and microservice architectures. *Journal of Systems and Software*, 159:110432, 2020.
- [87] Jinjin Lin, Pengfei Chen, and Zibin Zheng. Microscope: Pinpoint performance issues with causal graphs in micro-service environments. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11236 LNCS:3–20, 2018.
- [88] Xiao Zhang, Eric Tune, Robert Hagmann, Rohit Jnagal, Vrigo Gokhale, and John Wilkes. Cpi 2: Cpu performance isolation for shared compute clusters. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 379–391. ACM, 2013.
- [89] Artemiy Margaritov, Siddharth Gupta, Reikai Gonzalez-Alberquilla, and Boris Grot. Stretch: Balancing qos and throughput for colocated server workloads on smt cores. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 15–27. IEEE, 2019.
- [90] Liang Bao, Chase Wu, Xiaoxuan Bu, Nana Ren, and Mengqing Shen. Performance Modeling and Workflow Scheduling of Microservice-based Applications in Clouds. *IEEE Transactions on Parallel and Distributed Systems*, 2019.
- [91] Anshul Jindal, Vladimir Podolskiy, and Michael Gerndt. Performance Modeling for Cloud Microservice Applications.
- [92] Hamzeh Khazaei, Cornel Barna, and Marin Litoiu. Performance Modeling of Microservice Platforms Considering the Dynamics of the underlying Cloud Infrastructure. 2019.
- [93] Marco Gribaudo, Mauro Iacono, and Daniele Manini. Performance Evaluation Of Massively Distributed Microservices Based Applications. 6(Cd):598–604, 2017.
- [94] Ram Srivatsa Kannan, Lavanya Subramanian, Ashwin Raju, Jeongseob Ahn, Jason Mars, and Lingjia Tang. Grand slam: Guaranteeing slas for jobs in microservices execution frameworks. In *Proceedings of the Fourteenth EuroSys Conference 2019, EuroSys '19*, New York, NY, USA, 2019. Association for Computing Machinery.
- [95] Jaime Correia, Fabio Ribeiro, Ricardo Filipe, Filipe Araujo, and Jorge Cardoso. Response time characterization of microservice-based systems. *NCA 2018 - 2018 IEEE 17th International Symposium on Network Computing and Applications*, 2018.
- [96] Yinbo Yu, Jianfeng Yang, Chengcheng Guo, Hong Zheng, and Jiancheng He. Joint optimization of service request routing and instance placement in the microservice system. *Journal of Network and Computer Applications*, 147:102441, 2019.
- [97] Chengxin Yan, Ningjiang Chen, and Zhang Shuo. High-performance elastic management for cloud containers based on predictive message scheduling. *Future Internet*, 9(4), 2017.
- [98] Xiaofeng Hou, Jiacheng Liu, Chao Li, and Minyi Guo. Unleashing the scalability potential of power-constrained data center in the microservice era. In *Proceedings of the 48th International Conference on Parallel Processing, ICPP 2019*, New York, NY, USA, 2019. Association for Computing Machinery.
- [99] Carlos Guerrero, Isaac Lera, and Carlos Juiz. Resource optimization of container orchestration: a case study in multi-cloud microservices-based applications. *Journal of Supercomputing*, 74(7):2956–2983, 2018.
- [100] Xue Leng, Tzung-Han Juang, Yan Chen, and Han Liu. Aomo: An ai-aided optimizer for microservices orchestration. In *Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos*, pages 1–2, 2019.
- [101] Sander Klock, Jan Martijn E. M. Van Der Werf, Jan Pieter Guelen, and Slinger Jansen. Workload-Based Clustering of Coherent Feature Sets in Microservice Architectures. In *2017 IEEE International*

- Conference on Software Architecture (ICSA)*, pages 11–20. IEEE, apr 2017.
- [102] OpenStack. <https://www.openstack.org/>.
- [103] Kubernetes: Production-grade container orchestration. <https://kubernetes.io/>.
- [104] Davi Monteiro, Rômulo Gadelha, Paulo Henrique M. Maia, Lincoln S. Rocha, and Nabor C. Mendonça. Beethoven: An event-driven lightweight platform for microservice orchestration. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11048 LNCS:191–199, 2018.
- [105] Dong Guo, Wei Wang, Guosun Zeng, and Zerong Wei. Microservices Architecture Based Cloudware Deployment Platform for Service Computing. In *2016 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, pages 358–363. IEEE, mar 2016.
- [106] Joao Rufino, Muhammad Alam, Joaquim Ferreira, Abdur Rehman, and Kim Fung Tsang. Orchestration of containerized microservices for IIoT using Docker. In *Proceedings of the IEEE International Conference on Industrial Technology*, pages 1532–1536. IEEE, mar 2017.
- [107] Pieter J. Meulenhoff, Dennis R. Ostendorf, Miroslav Živković, Hendrik B. Meeuwissen, and Bart M. M. Gijzen. Intelligent overload control for composite web services. In Luciano Baresi, Chi-Hung Chi, and Jun Suzuki, editors, *Service-Oriented Computing*, pages 34–49, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.
- [108] Ben Christensen. Application resilience in a service-oriented architecture. June 10, 2013.
- [109] Matt Welsh, David Culler, and Eric Brewer. Seda: An architecture for well-conditioned, scalable internet services. In *PROC. SYMP. OPERATING SYSTEMS PRINCIPLES*, pages 230–243, 2001.
- [110] Hailong Yang, Alex Breslow, Jason Mars, and Lingjia Tang. Bubble-flux: Precise online qos management for increased utilization in warehouse scale computers. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, pages 607–618, New York, NY, USA, 2013. ACM.
- [111] Christina Delimitrou and Christos Kozyrakis. Quasar: resource-efficient and qos-aware cluster management. In *ACM SIGARCH Computer Architecture News*, volume 42, pages 127–144. ACM, 2014.
- [112] H. Alipour and Y. Liu. Online machine learning for cloud resource provisioning of microservice backend systems. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 2433–2441, Dec 2017.
- [113] Michael Alan Chang, Aurojit Panda, Yuan-Cheng Tsai, Hantao Wang, and Scott Shenker. Throttlebot - performance without insight. *CoRR*, abs/1711.00618, 2017.
- [114] Hao Zhou, Ming Chen, Qian Lin, Yong Wang, Xiaobin She, Sifan Liu, Rui Gu, Beng Chin Ooi, and Junfeng Yang. Overload control for scaling wechat microservices. In *Proceedings of the ACM Symposium on Cloud Computing*, SoCC '18, pages 149–161, New York, NY, USA, 2018. ACM.
- [115] Lalith Suresh, Peter Bodik, Ishai Menache, Marco Canini, and Florin Ciucu. Distributed resource management across process boundaries. pages 611–623, 2017.
- [116] M. Xu, A. N. Toosi, and R. Buyya. ibrownout: An integrated approach for managing energy and brownout in container-based clouds. *IEEE Transactions on Sustainable Computing*, 4(1):53–66, Jan 2019.
- [117] Chih-Hsun Chou, Laxmi N Bhuyan, and Daniel Wong.  $\mu$ dpm: Dynamic power management for the microsecond era. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 120–132. IEEE, 2019.
- [118] Amirhossein Mirhosseini, Akshitha Sriraman, and Thomas F Wenisch. Enhancing server efficiency in the face of killer microseconds. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 185–198. IEEE, 2019.
- [119] Harshad Kasture, Davide B Bartolini, Nathan Beckmann, and Daniel Sanchez. Rubik: Fast analytical power management for latency-critical systems. In *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 598–610. IEEE, 2015.

- [120] David Lo, Liqun Cheng, Rama Govindaraju, Luiz André Barroso, and Christos Kozyrakis. Towards energy proportionality for large-scale latency-critical workloads. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pages 301–312. IEEE, 2014.
- [121] Yanpei Liu, Stark C Draper, and Nam Sung Kim. Sleepscale: Runtime joint speed scaling and sleep states management for power efficient datacenters. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pages 313–324. IEEE, 2014.
- [122] Sol Boucher, Anuj Kalia, David G. Andersen, and Michael Kaminsky. Putting the "micro" back in microservice. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 645–650, Boston, MA, 2018. USENIX Association.
- [123] Edward Oakes, Leon Yang, Dennis Zhou, Kevin Houck, Tyler Harter, Andrea Arpaci-Dusseau, and Remzi Arpaci-Dusseau. {SOCK}: Rapid task provisioning with serverless-optimized containers. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*, pages 57–70, 2018.
- [124] Istemi Ekin Akkus, Ruichuan Chen, Ivica Rimac, Manuel Stein, Klaus Satzke, Andre Beck, Paarijaat Aditya, and Volker Hilt. Sand: Towards high-performance serverless computing. In *2018 Annual Technical Conference (ATC 18)*, pages 923–935, 2018.
- [125] Xiaohui Luo, Fengyuan Ren, and Tong Zhang. High performance userspace networking for containerized microservices. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11236 LNCS:57–72, 2018.