Front. Comput. Sci., 2022, 16(6): 166106
https://doi.org/10.1007/s11704-020-0072-3

REVIEW ARTICLE

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, Minyi GUO

Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China

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, Amazon, and Microsoft 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 [1].

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 [2,3], accelerators [4] and new server architecture [5,6]. Second, microservice-based applications have complex runtime dynamics. The service units are usually deployed in a distributed manner with lightweight virtualization. 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

Although 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. [7] presented the widely adopted architecture patterns and compared the advantages and disadvantages of these patterns. Alshuqayran et al. [8] 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. [9] investigated the security implications and solutions for cloud microservices. Yarygina et al. [10] 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. [11] investigated monolithic and microservice for Web applications in cloud, with an empahsis on the architecture patterns. Vural et al. [12] looked at the trend of microservices and discussed the research gaps as well as the differences between microservice and SOA architecture. Gouigoux et al. [13] also compared the architecture and designs in monolithic and microservice Web services. As for production systems, Di Francesco et al. [14] summarized a few key industrial practices of microservice systems.

Service management approach is an important direction.

Received February 19, 2020; accepted June 2, 2020

E-mail: chaol@sjtu.edu.cn

Manvi et al. [15] presented the researches on the resource management approaches and optimizations at the infrastructure layer. Vaquero et al. [16] addressed the service orchestration challenges upon the advent of various new technologies, such as edge computing, serverless computing and disaggregated datacenters. Pahl et al. [17] 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.

3) We summarize the open issues and challenges faced by cloud microservice management. We present the research opportunities from various aspects.

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, faced with the distributed nature of cloud system and applications. In general, microservice is defined as an architectural style for software design, as opposed to the “monolithic” style. 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. 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).

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.

Decentralized management Guided by the principle of “share as little as possible” microservices are typically self-managed and distributed among virtual or physical resources [18]. 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 [19]. 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 as shown in Table 1.

SOA Similar to microservices, Service-Oriented Architecture (SOA) features modularized design and message-based communication [20]. 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 [21]. 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

Table 1 Comparison of different computing paradigms

Paradigms	Virtualization level	Communication mode	Language diversity	Autonomous management	Stateless
Microservice	Container	Decentralized	Yes	Yes	Yes
Service oriented architecture	Bare-metal	Centralized	No	No	No
Serverless computing	Container/virtual machine	Decentralized/centralized	Yes	Yes	Yes
Event driven architecture	Virtual machine	Centralized	Yes	No	Yes
Model driven architecture	Not specified	Decentralized/centralized	No	No	Not specified

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. Generally it has two variants in the scenario of cloud platform: "Function as a Service" (FaaS) rather than "Backend as a Service" (BaaS). In serverless computing, 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 [22]. 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. 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 [23], serverless computing is suitable for short-running and event-driven jobs and services [16,24]. Microservice can be deployed in a serverless manner, and its difference from a serverless function, is that a microservice can consist of multiple functions. While there are many explorations to use serverless deployment for microservice in a cost-effective and scalable way [25,26], performance guarantee in this paradigm is a key research challenge [27].

Event driven architecture (EDA) The event-driven architecture is considered to be the foundation of the Microservice and FaaS [28]. 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 [29]. However, EDA brings challenges of the tracing [30] and performance guarantee [1,31] in distributed systems.

Model driven architecture (MDA) MDA is frequently proposed in microservice development [32,33]. It is firstly introduced as a model driven development approach [34], in which an abstraction of a system is used for designing a software application [35]. A model is a set of statements on system compositions and the relations with other systems [36]. In MDA, the microservice systems could be modeled by specific languages at different levels of abstractions. Meanwhile, the software implementations could be automatically generated by the description languages. Applying MDA in SOA has been addressed by prior researches [37,38]. According to prior work [33], 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 [1,39]. Although cloud applications are discomposed 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 Fig. 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.

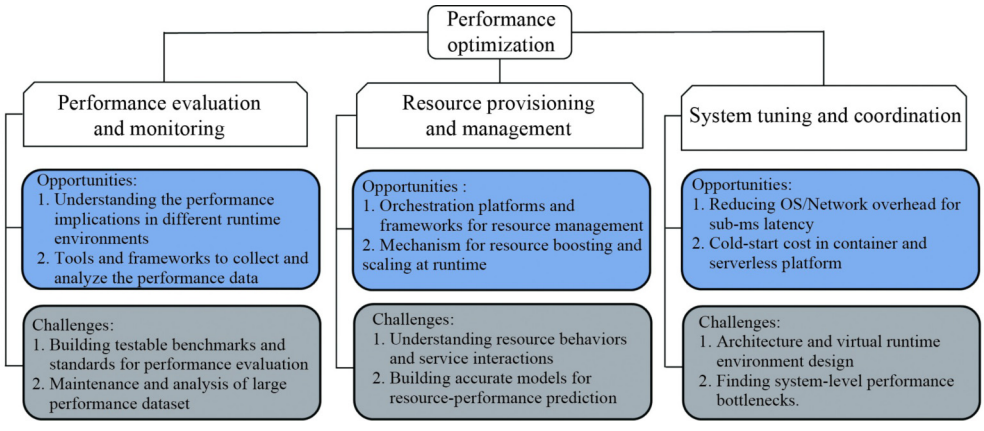


Fig. 1 Performance optimization techniques

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 Fig. 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. Table 2 shows the summary of prior works on performance evaluation and monitoring.

3.1 Benchmarking

Cloud service providers have developed simple open source benchmarks to facilitate microservice research. Since previous researches use simple benchmarks, their results are not validated in production environment. Therefore, more sophisticated benchmarks are needed [40]. 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 [41]. There are a few works that focus on the development and design of microservice system for various research purposes.

Aderaldo et al. [41] 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. [42] developed a medium-size benchmark TrainTicket, which covers three interaction modes and four programming languages. Gan et al. [43] 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 [44]. Kratzke et al. [45] first proposed a benchmark to study the fundamental design for microservice. Sriraman et al. [46] presented a benchmark: μSuite, 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. [47] 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.

Table 2 Summary of prior works on performance evaluation and monitoring

Categories	Subcategories	Prior work
Benchmarking	/	[40], [43], [44], [47], [41], [42],[45]
Analysis and characterization	System and architecture implications	[48], [49], [50], [51]
	Virtualization method evaluation	[52], [53], [54], [55], [56], [57], [58], [59], [60], [61]
	Application architecture evaluation	[7], [62], [46], [63], [18]
Monitoring and anomaly Detection	Tools & frameworks	[69], [1], [67], [68]
	Analysis & detection	[72], [73], [70], [71], [30]

System and architecture implications Ueda et al. [48] 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. [49] 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. [50] 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. [51] 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 comparision 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 [52]. Saha et al. [53] evaluated the performance in a Docker deployment and bare-metal deployment. Jaramillo et al. [54] presented a case study for Docker applied in the microservice architecture. The architecture patterns leveraging the VM, container, and serverless platforms are compared and key limitations of these technologies are detailed. Kang et al. [55] 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 [56] and conducted an multi-level analysis of cloud computing systems that leverage serverless and FaaS model. Esposito et al. [57] presented the designs of VM-based and container-based deployment and discussed concerns of different approaches. Villamizar et al. [58] 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. [59] conducted a study on the overhead and performance impact incurred by Docker Amaral et al. [60] 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. [61] 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. [7] 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 [62]. Saha et al. [53] evaluated the communication over virtual network in a containerized deployment for HPC workload. Sriraman et al. [46] 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. [63] 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. [18] identified the key factors considered by developers on the microservice granularity, and proposed some abstraction and evaluation approaches. Similarly, Lloyd et al. [64] 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 [65,66]. 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. [1] 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. [67] 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 [68] 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. [69] focused on the workflow-centric tracing and investigated the design space for tracing systems at

different layers. The authors pointed out that the tracing adoption should be reconsidered with the infrastructure and different management tasks for best utilities. They delved the implementations of existing tracing system, and conducted a systematic analysis on distributed services.

Performance anomaly detection and troubleshooting is another important research direction [70,71]. There are studies that use graph model in microservice system to detect the root cause of performance degradation [72]. It presents an analysis framework which can perform troubleshooting based on a library of anomalous graphs. Lin et al. [73] 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. [71] 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. [30] 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 in Table 3: 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 [74], and small fraction of peak loads [75]. Therefore, building accurate workload prediction model can be a challenging task.

Bao et al. [76] 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. [77] 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. [78] 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. [79] modeled the microservice system as an oriented graph. They used Montecarlo simulation to generate random application topologies. Then they derived the performance metrics for each generated scenario given VM allocation. Similarly, Kannan et al [80] proposed to model the microservice-based multi-stage jobs as Directed Acyclic Graph (DAG), and used the DAGs to estimate the completion time of requests.

Queuing model has also been used for microservice response time prediction. For example, Correia et al. [81] 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. [82] 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. [83] 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. [71] 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. [84] 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

Table 3 Summary of prior works on resource provisioning and management

Categories	Subcategories	Prior work
Modeling & prediction	Linear model	[76], [77]
	Non-Linear model	[81], [82], [78], [79], [80], [83], [71]
Placement and orchestration	Application layer	[85], [86], [82]
	Platform layer	[55], [88], [89], [90]
Runtime adaptation	Request scheduling	[83], [76], [80], [93], [94]
	Resource adjustment	[1], [51], [96], [31],[95]
	Overload control	[98], [99], [100]

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. [85] 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. [86] 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 [82] 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. [87] 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, Kubernetes, are widely adopted for service orchestration in large environments. Kang et al. [55] 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. [88] 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 [89,90].

4.3 Runtime adaptation

Performance optimization at runtime for microservices requires a mixture of advanced resource adaptation techniques.

Request scheduling Yan et al. [83] 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. [76] 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. [80] 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. [1] 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. Hou [95] proposed a learning-based resource management system that considers the dynamicity and heterogeneity of the microservice environment. The authors used a bipartite feature inference approach to extract the temporal characteristics of microservices. Liu [51] 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. [96] focused on the resource pattern for microservices and leveraged the machine learning for resource demand predication to mitigate low utilization. Cui et al. [31] focused on the asynchronous event-based service model. They propose to identify latency bottlenecks using event dependency graph and used an energy-efficient boosting technoqiue to optimize the tail latency. The techniques proposed by Chang et al. [97] 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 Overload control is another import aspect for services to maintain normal response or scale before saturation, including mechanisms such as load shedding [91] and resource boosting [92]. Zhou et al. [98] 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. [99] 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. [100] 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 [50]. Meanwhile, microservice applications have much shorter request service time, typically in the microsecond-scale [102]. 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. Table 4 summaries the prior works on system tuning and coordination.

Tail latency is the primal design focus of cloud datacenters in many prior works [46,102,103]. 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) [104–106]. However, 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. Hou et al. [101] proposed a necessary abstraction and optimization for highly efficient power management in the microservice environment. Chou et al. [102] 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 at al. [103] 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 to efficiently run on a scale-out architecture, thereby minimizing queuing delay and tail latency.

Since microservices typically have sub-microsecond latency latency requirement, the OS-level overhead——such as thread invoking and context switch, become non-trivial factors to tail latency. Sriraman et al. [46] 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 concurrency 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. [107] 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. [108] 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 attention as distributed microservices often run across multiple containers or even involves multi-server coordination. Akkus et al. [109] 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. [110] 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

Table 4 Summary of prior works on system tuning and coordination

Optimization target	Ref.	Methodology	Main results
Tail latency	[102]	CPU power state, Frequency scaling	Achieving stable tail latency, Saving CPU energy consumption
	[103]	Extending simultaneously multi-threaded cores to schedule short requests	Reducing tail latency under moderate and high load
	[46]	Automatically tuning the threading model for mid-tier microservice	1.9 × tail latency speedup over static threading choices
Invocation latency	[107]	Using language-level isolation and fast preemption	Achieving μs scale latency
	[108]	Using lightweight container isolation and package-aware caching	Achieving significant speedup over Docker
Communication overhead	[109]	Using application sandboxing and a hierarchical message bus	Achieving 43 speedup over OpenWhisk
	[110]	Using a userspace networking stack	Achieving high throughput and low latency

networking stack that uses rate limiters for performance isolation between different containers. It constructs fast channels between partner microservices 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 Fig. 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 dependencies, 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 μs-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 μs-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

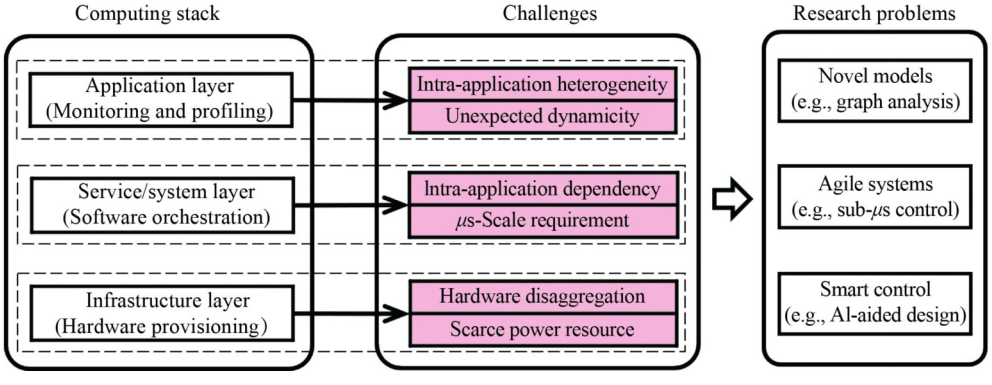


Fig. 2 Open issues in the microservice era

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 This work was sponsored by the National Natural Science Foundation of China (Grant No. 61972247). Corresponding author is Chao Li from Shanghai Jiao Tong University, China.

References

1. Gan Y, Zhang Y, Hu K, Cheng D, He Y, Pancholi M, Delimitrou C. Seer: leveraging big data to navigate the complexity of performance debugging in cloud microservices. In: Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems. 2019, 19–33
2. Chen Y, Luo T, Liu S, Zhang S, He L, Wang J, Li L, Chen T, Xu Z, Sun N, Temam O. DaDianNao: a machine-learning supercomputer. In: Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture. 2014, 609–622
3. Jouppi N P, Young C, Patil N, Patterson D, Agrawal G, et al. In-datacenter performance analysis of a tensor processing unit. In: Proceedings of the 44th Annual International Symposium on Computer Architecture. 2017, 1–12
4. Chung E, Fowers J, Ovtcharov K, Papamichael M, Caulfield A, et al. Serving DNNs in real time at datacenter scale with project brainwave. IEEE Micro, 2018, 38(2): 8–20
5. Nitu V, Teabe B, Tchana A, Isci C, Hagimont D. Welcome to zombieland: practical and energy-efficient memory disaggregation in a datacenter. In: Proceedings of the 13th EuroSys Conference. 2018, 16
6. Lim K, Chang J, Mudge T, Ranganathan P, Reinhardt S K, Wenisch T. Disaggregated memory for expansion and sharing in blade servers. In: Proceedings of the 36th Annual International Symposium on Computer Architecture. 2009, 267–278
7. Taibi D, Lenarduzzi V, Pahl C. Architectural patterns for microservices: a systematic mapping study. In: Proceedings of the 8th International Conference on Cloud Computing and Services Science. 2018, 221–232
8. Alshuqayran N, Ali N, Evans R. A systematic mapping study in microservice architecture. In: Proceedings of the 9th IEEE International Conference on Service-Oriented Computing and Applications. 2016, 44–51
9. Aguiar L, Almeida W, Hazin R, Lima A, Ferraz F. Survey on microservice architecture-security, privacy and standardization on cloud computing environment. In: Proceedings of the 12th International Conference on Software Engineering Advances. 2017, 210
10. Yarygina T, Bagge A B. Overcoming security challenges in microservice architectures. In: Proceedings of 2018 IEEE Symposium on Service-Oriented System Engineering. 2018, 11–20
11. Villamizar M, Garcés O, Castro H, Verano M, Salamanca L, Casallas R, Gil S. Evaluating the monolithic and the microservice architecture pattern to deploy Web applications in the cloud. In: Proceedings of the 10th Computing Colombian Conference. 2015, 583–590
12. Vural H, Koyuncu M, Guney S. A systematic literature review on microservices. In: Proceedings of the 17th International Conference on Computational Science and its Applications. 2017, 203–217
13. Gouigoux J P, Tamzalit D. From monolith to microservices: lessons learned on an industrial migration to a Web oriented architecture. In: Proceedings of 2017 IEEE International Conference on Software Architecture Workshops. 2017, 62–65
14. Di Francesco P, Lago P, Malavolta I. Migrating towards microservice architectures: an industrial survey. In: Proceedings of 2018 IEEE International Conference on Software Architecture. 2018, 29–2909
15. Manvi S S, Shyam G K. Resource management for Infrastructure as a Service (IaaS) in cloud computing: a survey. Journal of Network and Computer Applications, 2014, 41: 424–440
16. Vaquero L M, Cuadrado F, Elkhatib Y, Bernal-Bernabe J, Srirama S N, Zhani M F. Research challenges in nextgen service orchestration. Future Generation Computer Systems, 2019, 90: 20–38
17. Pahl C, Jamshidi P. Microservices: a systematic mapping study. In: Proceedings of the 6th International Conference on Cloud Computing and Services Science. 2016, 137–146
18. Hassan S, Bahsoon R. Microservices and their design trade-offs: a self-adaptive roadmap. In: Proceedings of 2016 IEEE International Conference on Services Computing. 2016, 813–818
19. Toffetti G, Brunner S, Blöchlinger M, Dudouet F, Edmonds A. An architecture for self-managing microservices. In: Proceedings of the 1st International Workshop on Automated Incident Management in Cloud. 2015, 19–24
20. Familiar B. Microservices, IoT, and Azure: Leveraging DevOps and Microservice Architecture to Deliver SaaS Solutions. Berkeley: Apress, 2015
21. Jamshidi P, Pahl C, Mendonça N C, Lewis J, Tilkov S. Microservices: the journey so far and challenges ahead. IEEE Software, 2018, 35(3): 24–35
22. Baldini I, Castro P, Chang K, Cheng P, Fink S, Ishakian V, Mitchell N, Muthusamy V, Rabbah R, Slominski A, Suter P. Serverless computing: current trends and open problems. In: Chaudhary S, Somani G, Buyya R, eds. Research Advances in Cloud Computing. Singapore: Springer, 2017, 1–20
23. Fox G C, Ishakian V, Muthusamy V, Slominski A. Status of serverless computing and function-as-a-service (FaaS) in industry and research. 2017, arXiv preprint arXiv: 1708.08028
24. Castro P, Ishakian V, Muthusamy V, Slominski A. Serverless programming (function as a service). In: Proceedings of the IEEE 37th International Conference on Distributed Computing Systems. 2017, 2658−2659
25. Yan M, Castro P, Cheng P, Ishakian V. Building a chatbot with serverless computing. In: Proceedings of the 1st International Workshop on Mashups of Things and APIs. 2016, 5
26. Ishakian V, Muthusamy V, Slominski A. Serving deep learning models in a serverless platform. In: Proceedings of 2018 IEEE International Conference on Cloud Engineering. 2018, 257–262
27. Castro P, Ishakian V, Muthusamy V, Slominski A. The rise of serverless computing. Communications of the ACM, 2019, 62(12): 44–54
28. Kritikos K, Skrzypek P. A review of serverless frameworks. In: Proceedings of IEEE/ACM International Conference on Utility and Cloud Computing Companion. 2018, 161–168
29. Michelson B M. Event-driven architecture overview. Patricia Seybold Group, 2006, 2(12): 10–1571
30. Thalheim J, Rodrigues A, Akkus I E, Bhatotia P, Chen R, Viswanath B, Jiao L, Fetzer C. Sieve: actionable insights from monitored metrics in distributed systems. In: Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference. 2017, 14–27
31. Cui W, Richins D, Zhu Y, Reddi V J. 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. 2019, 152–164
32. Terzic B, Dimitrieski V, Kordić S, Luković I. A model-driven approach to microservice software architecture establishment. In: Proceedings of 2018 Federated Conference on Computer Science and Information Systems. 2018, 73–80
33. Rademacher F, Sachweh S, Zündorf A. Differences between model-

driven development of service-oriented and microservice architecture. In: Proceedings of 2017 IEEE International Conference on Software Architecture Workshops. 2017, 38–45
34. Mellor S, Scott K, Uhl A, Weise D. Model-driven architecture. In: Proceedings of International Conference on Object-Oriented Information Systems. 2002, 290–297
35. Da Silva A R. Model-driven engineering: a survey supported by the unified conceptual model. Computer Languages, Systems & Structures, 2015, 43: 139–155
36. Seidewitz E. What models mean. IEEE Software, 2003, 20(5): 26–32
37. Vale S, Hammoudi S. Model driven development of context-aware service oriented architecture. In: Proceedings of the 11th IEEE International Conference on Computational Science and Engineering-Workshops. 2008, 412–418
38. Ameller D, Burgués X, Collell O, Costal D, Franch X, Papazoglou M P. Development of service-oriented architectures using model-driven development: a mapping study. Information and Software Technology, 2015, 62: 42–66
39. Fazio M, Celesti A, Ranjan R, Liu C, Chen L, Villari M. Open issues in scheduling microservices in the cloud. IEEE Cloud Computing, 2016, 3(5): 81–88
40. Zhou X, Peng X, Xie T, Sun J, Xu C, Ji C, Zhao W. Benchmarking microservice systems for software engineering research. In: Proceedings of the 40th IEEE/ACM International Conference on Software Engineering: Companion. 2018, 323–324
41. Aderaldo C M, Mendonça N C, Pahl C, Jamshidi P. Benchmark requirements for microservices architecture research. In: Proceedings of the 1st IEEE/ACM International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering. 2017, 8–13
42. Zhou X, Peng X, Xie T, Sun J, Ji C, Li W, Ding D. Fault analysis and debugging of microservice systems: industrial survey, benchmark system, and empirical study. IEEE Transactions on Software Engineering, 2021, 47(2): 243–260
43. Gan Y, Zhang Y, Cheng D, Shetty A, Rathi P, et al. An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems. In: Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems. 2019, 3−18
44. Sriraman A, Wenisch T F. μ suite: a benchmark suite for microservices. In: Proceedings of 2018 IEEE International Symposium on Workload Characterization. 2018, 1−12
45. Kratzke N, Quint P C. ppbench-a visualizing network benchmark for microservices. In: Proceedings of the 6th International Conference on Cloud Computing and Services Science. 2016, 223−231
46. Sriraman A, Wenisch T F. μtune: auto-tuned threading for OLDI microservices. In: Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation. 2018, 177−194
47. Papapanagiotou I, Chella V. NDBench: benchmarking microservices at scale. 2018, arXiv preprint arXiv: 1807.10792
48. Ueda T, Nakaike T, Ohara M. Workload characterization for microservices. In: Proceedings of 2016 IEEE International Symposium on Workload Characterization. 2016, 1−10
49. Gan Y, Delimitrou C. The architectural implications of cloud microservices. IEEE Computer Architecture Letters, 2018, 17(2): 155–158
50. Sriraman A, Dhanotia A, Wenisch T F. SoftSKU: optimizing server architectures for microservice diversity @scale. In: Proceedings of the 46th International Symposium on Computer Architecture. 2019, 513−526
51. Liu L. Qos-aware machine learning-based multiple resources scheduling for microservices in cloud environment. 2019, arXiv preprint arXiv: 1911.13208
52. Xavier B, Ferreto T, Jersak L. Time provisioning evaluation of KVM, docker and unikernels in a cloud platform. In: Proceedings of the 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. 2016, 277−280
53. Saha P, Beltre A, Uminski P, Govindaraju M. Evaluation of docker containers for scientific workloads in the cloud. In: Proceedings of the Practice and Experience on Advanced Research Computing. 2018, 11
54. Jaramillo D, Nguyen D V, Smart R. Leveraging microservices architecture by using docker technology. In: Proceedings of 2016 IEEE SoutheastCon. 2016, 1−5
55. Kang H, Le M, Tao S. Container and microservice driven design for cloud infrastructure DevOps. In: Proceedings of 2016 IEEE International Conference on Cloud Engineering. 2016, 202−211
56. Lynn T, Rosati P, Lejeune A, Emeakaroha V. A preliminary review of enterprise serverless cloud computing (function-as-a-service) platforms. In: Proceedings of 2017 IEEE International Conference on Cloud Computing Technology and Science. 2017, 162−169
57. Esposito C, Castiglione A, Choo K K R. Challenges in delivering software in the cloud as microservices. IEEE Cloud Computing, 2016, 3(5): 10–14
58. Villamizar M, Garcés O, Ochoa L, Castro H, Salamanca L, Verano M, Casallas R, Gil S, Valencia C, Zambrano A, Lang M. Infrastructure cost comparison of running Web applications in the cloud using AWS lambda and monolithic and microservice architectures. In: Proceedings of the 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. 2016, 179−182
59. Friðriksson V. Container overhead in microservice systems. KTH Royal Institute of Technology, Dissertation, 2018
60. Amaral M, Polo J, Carrera D, Mohomed I, Unuvar M, Steinder M. Performance evaluation of microservices architectures using containers. In: Proceedings of the 14th IEEE International Symposium on Network Computing and Applications. 2015, 27−34
61. Kratzke N. About microservices, containers and their underestimated impact on network performance. 2017, arXiv preprint arXiv: 1710.04049
62. Osses F, Márquez G, Astudillo H. Poster: exploration of academic and industrial evidence about architectural tactics and patterns in microservices. In: Proceedings of the 40th IEEE/ACM International Conference on Software Engineering: Companion. 2018, 256−257
63. Shadija D, Rezai M, Hill R. Microservices: granularity vs. performance. In: Proceedings of the10th International Conference on Utility and Cloud Computing. 2017, 215−220
64. Lloyd W, Ramesh S, Chinthalapati S, Ly L, Pallickara S. Serverless computing: an investigation of factors influencing microservice performance. In: Proceedings of 2018 IEEE International Conference on Cloud Engineering. 2018, 159−169
65. Baek H, Srivastava A, Van der Merwe J. 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. 2017, 268−273
66. Da Cunha Rodrigues G, Calheiros R N, Guimaraes V T, Dos Santos G L, De Carvalho M B, Granville L, Tarouco L M R, Buyya R. Monitoring of cloud computing environments: concepts, solutions, trends, and future directions. In: Proceedings of the 31st Annual ACM Symposium on Applied Computing. 2016, 378−383
67. Nicol J, Li C, Chen P, Feng T, Ramachandra H. ODP: an infrastructure for on-demand service profiling. In: Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering. 2018, 139−144
68. Cinque M, Della Corte R, Pecchia A. Microservices monitoring with event logs and black box execution tracing. IEEE Transactions on Services Computing, 2019, DOI: 10.1109/TSC.2019.2940009
69. Sambasivan R, Shafer I, Mace J, Sigelman B, Fonseca R, Ganger G R. Principled workflow-centric tracing of distributed systems. In: Proceedings of the 7th ACM Symposium on Cloud Computing. 2016,

401−414

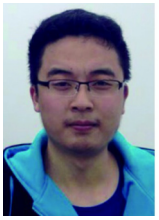
70. Wu L, Bogatinovski J, Nedelkoski S, Tordsson J, Kao O. Performance diagnosis in cloud microservices using deep learning. In: Proceedings of International Conference on Service-Oriented Computing. 2020, 85–96
71. Ravichandiran R, Bannazadeh H, Leon-Garcia A. Anomaly detection using resource behaviour analysis for Autoscaling systems. In: Proceedings of the 4th IEEE Conference on Network Softwarization and Workshops. 2018, 192−196
72. Brandón Á, Solé M, Huélamo A, Solans D, Pérez M S, Muntés-Mulero V. Graph-based root cause analysis for service-oriented and microservice architectures. Journal of Systems and Software, 2020, 159: 110432
73. Lin J, Chen P, Zheng Z. Microscope: pinpoint performance issues with causal graphs in micro-service environments. In: Proceedings of the 16th International Conference on Service-Oriented Computing. 2018, 3−20
74. Zhang X, Tune E, Hagmann R, Jnagal R, Gokhale V, Wilkes J. CPI^2: CPU performance isolation for shared compute clusters. In: Proceedings of the 8th ACM European Conference on Computer Systems. 2013, 379−391
75. Margaritov A, Gupta S, Gonzalez-Alberquilla R, Grot B. Stretch: balancing QoS and throughput for colocated server workloads on SMT cores. In: Proceedings of 2019 IEEE International Symposium on High Performance Computer Architecture. 2019, 15−27
76. Bao L, Wu C, Bu X, Ren N, Shen M. Performance modeling and workflow scheduling of microservice-based applications in clouds. IEEE Transactions on Parallel and Distributed Systems, 2019, 30(9): 2114–2129
77. Jindal A, Podolskiy V, Gerndt M. Performance modeling for cloud microservice applications. In: Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering. 2019, 25−32
78. Khazaei H, Mahmoudi N, Barna C, Litoiu M. Performance modeling of microservice platforms. 2019, arXiv preprint arXiv: 1902.0338
79. Gribaudo M, Iacono M, Manini D. Performance evaluation of massively distributed microservices based applications. In: Proceedings of the 31st European Conference on Modelling and Simulation. 2017, 598−604
80. Kannan R S, Subramanian L, Raju A, Ahn J, Mars J, Tang L. GrandSLAm: guaranteeing SLAs for jobs in microservices execution frameworks. In: Proceedings of the 14th EuroSys Conference. 2019, 34
81. Correia J, Ribeiro F, Filipe R, Arauio F, Cardoso J. Response time characterization of microservice-based systems. In: Proceedings of the 17th IEEE International Symposium on Network Computing and Applications. 2018, 1−5
82. Yu Y, Yang J, Guo C, Zheng H, He J. Joint optimization of service request routing and instance placement in the microservice system. Journal of Network and Computer Applications, 2019, 147: 102441
83. Yan C, Chen N, Shuo Z. High-performance elastic management for cloud containers based on predictive message scheduling. Future Internet, 2017, 9(4): 87
84. Hou X, Liu J, Li C, Guo M. Unleashing the scalability potential of power-constrained data center in the microservice era. In: Proceedings of the 48th International Conference on Parallel Processing. 2019, 10
85. Guerrero C, Lera I, Juiz C. Resource optimization of container orchestration: a case study in multi-cloud microservices-based applications. The Journal of Supercomputing, 2018, 74(7): 2956–2983
86. Leng X, Juang T H, Chen Y, Liu H. AOMO: an AI-aided optimizer for microservices orchestration. In: Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos. 2019, 1−2
87. Klock S, van der Werf J M E M, Guelen J P, Jansen S. Workload-based clustering of coherent feature sets in microservice architectures. In: Proceedings of 2017 IEEE International Conference on Software Architecture. 2017, 11−20
88. Monteiro D, Gadelha R, Maia P H M, Rocha L S, Mendonça N C. Beethoven: an event-driven lightweight platform for microservice orchestration. In: Proceedings of the 12th European Conference on Software Architecture. 2018, 191−199
89. Guo D, Wang W, Zeng G, Wei Z. Microservices architecture based cloudware deployment platform for service computing. In: Proceedings of 2016 IEEE Symposium on Service-Oriented System Engineering. 2016, 358−363
90. Rufino J, Alam M, Ferreira J, Rehman A, Tsang K F. Orchestration of containerized microservices for IIoT using Docker. In: Proceedings of 2017 IEEE International Conference on Industrial Technology. 2017, 1532−1536
91. Meulenhoff P J, Ostendorf D R, Živković M, Meeuwissen H B, Gijsen B M M. Intelligent overload control for composite web services. In: Proceedings of the 7th International Joint Conference on Service-Oriented Computing. 2009, 34−49
92. Welsh M, Culler D, Brewer E. SEDA: an architecture for well-conditioned, scalable internet services. In: Proceedings of the 18th ACM Symposium on Operating Systems Principles. 2001, 230−243
93. Yang H, Breslow A, Mars J, Tang L. Bubble-flux: precise online QoS management for increased utilization in warehouse scale computers. In: Proceedings of the 40th Annual International Symposium on Computer Architecture. 2013, 607−618
94. Delimitrou C, Kozyrakis C. Quasar: resource-efficient and QoS-aware cluster management. In: Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems. 2014, 127−144
95. Hou X, Li C, Liu J, Zhang L, Ren S, Leng J, Chen Q, Guo M. AlphaR: learning-powered resource management for irregular, dynamic microservice graph. In: Proceedings of 2021 IEEE International Parallel and Distributed Processing Symposium. 2021, 797−806
96. Alipour H, Liu Y. Online machine learning for cloud resource provisioning of microservice backend systems. In: Proceedings of 2017 IEEE International Conference on Big Data. 2017, 2433−2441
97. Chang M A, Panda A, Tsai Y C, Wang H, Shenker S. ThrottleBot - performance without insight. 2017, arXiv preprint arXiv: 1711.00618
98. Zhou H, Chen M, Lin Q, Wang Y, She X, Liu S, Gu R, Ooi B C, Yang J. Overload control for scaling WeChat microservices. In: Proceedings of the ACM Symposium on Cloud Computing. 2018, 149−161
99. Suresh L, Bodik P, Menache I, Canini M, Ciucu F. Distributed resource management across process boundaries. In: Proceedings of 2017 Symposium on Cloud Computing. 2017, 611−623
100. Xu M, Toosi A N, Buyya R. iBrownout: an integrated approach for managing energy and brownout in container-based clouds. IEEE Transactions on Sustainable Computing, 2019, 4(1): 53–66
101. Hou X, Li C, Liu J, Zhang L, Hu Y, Guo M. ANT-man: towards agile power management in the microservice era. In: Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis. 2020, 78
102. Chou C H, Bhuyan L N, Wong D. μDPM: dynamic power management for the microsecond era. In: Proceedings of 2019 International Symposium on High Performance Computer Architecture. 2019, 120−132
103. Mirhosseini A, Sriraman A, Wenisch T F. Enhancing server efficiency in the face of killer microseconds. In: Proceedings of 2019 IEEE International Symposium on High Performance Computer Architecture. 2019, 185−198
104. Kasture H, Bartolini D B, Beckmann N, Sanchez D. Rubik: fast analytical power management for latency-critical systems. In: Proceedings of the 48th Annual IEEE/ACM International Symposium on Microarchitecture. 2015, 598−610
105. Lo D, Cheng L, Govindaraju R, Barroso L A, Kozyrakis C. Towards energy proportionality for large-scale latency-critical workloads. In:

Proceedings of the 41st ACM/IEEE International Symposium on Computer Architecture. 2014, 301−312

106. Liu Y, Draper S C, Kim N S. SleepScale: runtime joint speed scaling and sleep states management for power efficient data centers. In: Proceedings of the 41st Annual International Symposium on Computer Architecuture. 2014, 313−324
107. Boucher S, Kalia A, Andersen D G, Kaminsky M. Putting the “micro” back in microservice. In: Proceedings of 2018 USENIX Annual Technical Conference. 2018, 645−650
108. Oakes E, Yang L, Zhou D, Houck K, Harter T, Arpaci-Dusseau A, Arpaci-Dusseau R H. SOCK: rapid task provisioning with serverless-optimized containers. In: Proceedings of 2018 USENIX Annual Technical Conference. 2018, 57−69
109. Akkus I, Chen R, Rimac I, Stein M, Satzke K, Beck A, Aditya P, Hilt V. SAND: towards high-performance serverless computing. In: Proceedings of 2018 USENIX Conference on Usenix Annual Technical Conference. 2018, 923−935
110. Luo X, Ren F, Zhang T. High performance userspace networking for containerized microservices. In: Proceedings of the 16th International Conference on Service-Oriented Computing. 2018, 57−72

Rong Zeng received her MS degree from the Department of Computer Science and Engineering, Shanghai Jiao Tong University, China in 2020 and BS degree from the College of Computer Science and Technology, Jilin University, China. Her research interests include higher-performance and resource-efficient computing in datacenters.

Xiaofeng Hou received her PhD degree from the Department of Computer Science and Engineering, Shanghai Jiao Tong University, China in 2020 and BS degree from Dalian University of Technology, China. She is currently a Postdoctoral Researcher in The Hong Kong University of Science and Technology, China. Her main research interests include computer architecture and highly available datacenter systems. She has received a Best Paper Award from ICCD in 2018.



Lu Zhang received the BS degree from the Northwestern Polytechnical University, China in 2016. He is working toward the PhD degree in the Department of Computer Science and Engineering, Shanghai Jiao Tong University, China. His research interests include edge computing, network function virtualization and serverless computing.

Chao Li received his PhD degree from the University of Florida, USA in 2014. He is currently an associate professor with tenure in the Department of Computer Science and Engineering, Shanghai Jiao Tong University, China. His research mainly focuses on computer architecture and systems for emerging applications. He is a senior member of IEEE/ACM/CCF.

Wenli Zheng received his PhD degree from the Ohio State University, USA in 2016. He is currently an assistant professor in the Department of Computer Science and Engineering, Shanghai Jiao Tong University, China. His research interests include large scale energy efficient computing system, cooperative computing and trusted computing.

Minyi Guo received his PhD degree in computer science from the University of Tsukuba, Japan. He is currently Zhiyuan Chair professor, Shanghai Jiao Tong University, China. His research interests include parallel/distributed computing, compiler optimizations, cloud computing and big data. He is now on the editorial board of IEEE Transactions on Parallel and Distributed Systems and Journal of Parallel and Distributed Computing. Dr. Guo is IEEE fellow and CCF fellow.