Placement of Microservices-based IoT Applications in Fog Computing: A Taxonomy and Future Directions

SAMODHA PALLEWATTA, VASSILIS KOSTAKOS, and RAJKUMAR BUYYA, The Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, University of Melbourne, Australia

The Fog computing paradigm utilises distributed, heterogeneous and resource-constrained devices at the edge of the network for efficient deployment of latency-critical and bandwidth-hungry IoT application services. Moreover, MicroService Architecture (MSA) is increasingly adopted to keep up with the rapid development and deployment needs of fast-evolving IoT applications. Due to the fine-grained modularity of the microservices and their independently deployable and scalable nature, MSA exhibits great potential in harnessing Fog and Cloud resources, thus giving rise to novel paradigms like Osmotic computing. The loosely coupled nature of the microservices, aided by the container orchestrators and service mesh technologies, enables the dynamic composition of distributed and scalable microservices to achieve diverse performance requirements of the IoT applications using distributed Fog resources. To this end, efficient placement of microservice plays a vital role, and scalable placement algorithms are required to utilise the said characteristics of the MSA while overcoming novel challenges introduced by the architecture. Thus, we present a comprehensive taxonomy of recent literature on microservices-based IoT applications placement within Fog computing environments. Furthermore, we organise multiple taxonomies to capture the main aspects of the placement problem, analyse and classify related works, identify research gaps within each category, and discuss future research directions.

CCS Concepts: • **General and reference** → **General literature**; • **Computer systems organization** → **Distributed architectures**;

Additional Key Words and Phrases: Fog computing, microservice architecture, Internet of Things, Osmotic computing, application placement

1 INTRODUCTION

The **Internet of Things (IoT)** paradigm is gaining immense popularity due to its significance in technical, social, and economic aspects [9]. IoT transforms everyday objects and infrastructure into intelligent entities that can interact with each other without human intervention, which has resulted in its expansion to a wide range of services, including healthcare, transportation, industrialisation, agriculture, and so on. IoT generates enormous quantities of dynamic data composed of various data types to be processed, analysed, and stored. Cloud computing was initially identified as a viable solution for hosting such IoT services, giving rise to cloud-centric IoT [42]. However, due to the exponential increase in connected devices, raw data transmission towards centralised Cloud data centres increases network congestion and latency, thus reducing the feasibility of cloud-centric IoT analytics. As a solution, distributed computing paradigms such as Fog computing, Edge computing, and **Mobile Edge Computing (MEC)** are introduced, bringing data processing and storage closer to the end-user, hence supporting latency-critical and bandwidth-consuming IoT services.

The Fog computing paradigm, first introduced by Cisco in 2012, uses compute, storage, and networking services residing in the intermediate layer between IoT devices and the Cloud (e.g., smart routers and switches, micro datacentres, small-cell base stations, nano servers, Raspberry Pi devices, cloudlets, etc.) [10]. These Fog devices are heterogeneous, resource-constrained, and geo-distributed compared to Cloud data centres. Thus, IoT application placement within Fog environments has to consider above characteristics of Fog devices and efficiently utilise them to ensure the performance requirements of IoT application services. Compared to Fog computing, Edge computing considers only the resources located exclusively at the edge of the network (e.g., mobile phones, access points, etc.) [41], whereas MEC focuses on the edge of the mobile network by improving the capabilities of the base stations located within the **Radio Access Network (RAN)** of the 5G/6G networks [20]. Thus, Edge computing and MEC are limited to the resources in the outer layer of the hierarchical Fog layer [55], while Fog computing expands further to include resources in multiple hierarchical levels extending towards Cloud and the federation among those resources to support IoT application workloads (although some research works use them interchangeably [16, 21, 90]). However, all three paradigms share common characteristics, such as proximity to the network edge, low-latency support, location awareness, geo-distribution, and integration with Cloud to overcome resource limitations.

Meanwhile, **MicroService Architecture (MSA)** emerged as a cloud-native application architecture style, enabling the development and deployment of highly reliable and scalable software systems that can undergo frequent updates and deployments [89, 104]. This paved the way for the convergence of Fog computing, IoT, and microservices, thus resulting in the introduction of novel paradigms such as Osmotic Computing that focus on dynamic placement and deployment of microservices across federated Fog-Cloud environments. Microservices-based IoT applications are gaining tremendous momentum due to their potential to improve the performance of IoT services deployed within distributed computing environments [73]. Compared to previous application architectures such as monolithic architecture and **Service Oriented Architecture (SOA)** realised through web services, the true potential of MSA as a cloud-native application architecture lies in its loosely coupled nature, which enables containerised deployment, dynamic composition, and load-balancing across federated multi-fog and multi-cloud environments with the support of other cloud-native technologies such as container orchestrators (e.g., Kubernetes, Docker Swarm) and service mesh technologies (e.g., Istio, Linkerd). Microservices can be independently and dynamically deployed and horizontally scaled across hybrid environments while maintaining seamless connectivity among interacting microservices under dynamic conditions. Thus, within geo-distributed and heterogeneous Fog environments, the placement of microservices-based
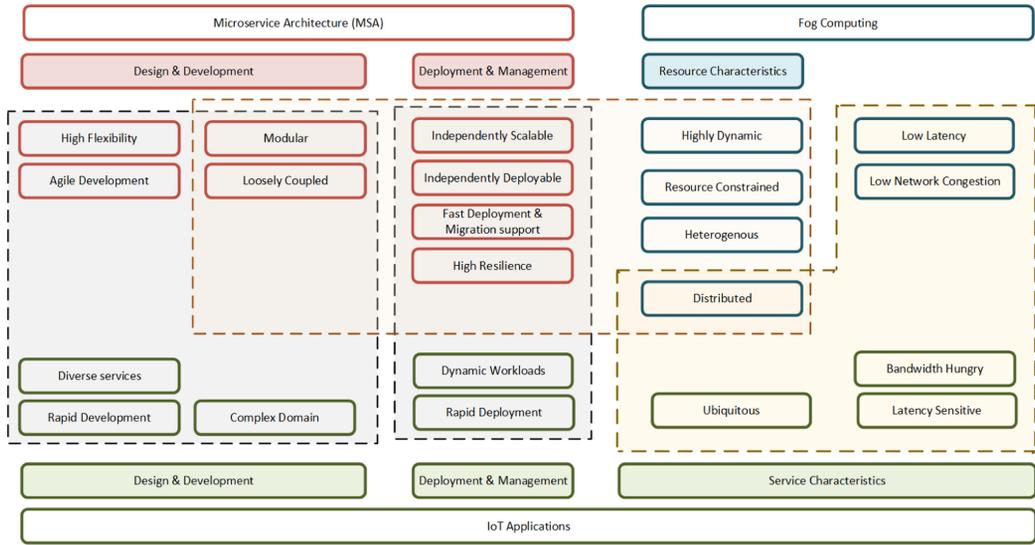
Fig. 1. IoT Applications, Microservice Architecture, and Fog computing.

applications remains one of the most critical and challenging areas. The placement algorithms benefit from awareness of the MSA-related characteristics so they can adequately utilise the strengths of the application architecture while overcoming its challenges. To this end, we aim to understand the features and challenges related to the development of placement policies for microservices-based IoT applications within Fog environments and analyse recent literature to summarise the current status of the research and propose possible future research directions.

1.1 Microservices-based IoT Applications and Fog Computing

This section highlights how the main characteristics of MSA, IoT applications, and Fog computing paradigm fit perfectly together, giving rise to microservices-based IoT applications for Fog environments. Figure 1 lists these characteristics and groups them to show their relationships.

Characteristics of IoT Applications can be categorised under three main groups; Design and Development, Deployment and Management, and Service Characteristics. Rapid design, development needs and interoperability, and service reusability caused by the IoT ecosystem's complexity can be satisfied by adopting a software architecture that supports higher flexibility to change, reduced time to market, and collaboration among multiple development teams. MSA can enable these requirements with the fine-grained modular design and loosely coupled nature that perfectly conforms with agile design and development principles. From a deployment and maintenance perspective, IoT applications must maintain rapid deployment cycles with minimum service disruptions and support the dynamic workload while maintaining service availability and resilience. These requirements can be successfully satisfied through the use of MSA [3, 14, 92]. MSA decomposes large and complicated applications into independently deployable and scalable modules that can be conveniently packaged using container technologies such as Docker and rktlet. Lightweight containers with considerably lower startup times enable quick deployment and migration, which result in higher service availability and reliability under dynamic conditions. The decomposition of the application into small independent units allows only the updated or newly developed microservices to be re-deployed and just the performance-affected microservices to be scaled within each deployment cycle. Moreover, this enables a proper balance between horizontal and vertical

scalability where microservices that are harder to scale horizontally (i.e., services that use relational databases, etc.) can be vertically scaled while the rest can be horizontally scaled [66]. Thus, MSA meets the scalability, maintainability, extensibility, and interoperability requirements of large and complex IoT software systems [84]. As a result, MSA is increasingly adopted for IoT application development in many areas such as smart cities, smart healthcare, IIoT [13, 22, 97].

Radio access network (RAN) technologies are improving rapidly to support higher bandwidth (i.e., 5G, 6G, etc.) and lower latency values. However, core network capabilities are still limited, thus making cloud-centric IoT infeasible to support IoT workloads. Sending a large amount of data towards the centralised Cloud through the core network would result in higher network congestion and larger latency values, lowering the QoS satisfaction of the latency-critical IoT services. Fog computing emerges as the solution to these challenges, enabling the data analytics closer to the end devices that generate the data, thus reducing the amount of data sent towards the Cloud. Federation of Fog environments with Cloud data centres allows IoT applications to utilise the best of both paradigms. This enables latency-critical, bandwidth-hungry services to be placed closer to the end-user while the rest can use Cloud resources, thus meeting heterogeneous QoS requirements of the IoT application services. Moreover, the distributed Fog architecture creates location awareness, thus providing ubiquitous access to IoT services. With Fog resources becoming richer in resource availability and telecommunication providers supporting Fog implementation, Fog computing is rising in popularity as a viable solution for hosting IoT applications.

In contrast to Cloud data centres, Fog environments consist of distributed, heterogeneous, resource-constrained devices. Thus, deploying a monolithic application onto such devices is less feasible due to their resource demand. The scalability of such applications is also limited by the said characteristic of Fog devices. Fine-grained microservices match such environments better due to their modular, loosely coupled nature, which makes the resource requirements of each microservice small enough to be satisfied by the distributed resources. Independent deployability and scalability of such microservices enable them to adjust to dynamic conditions (i.e., device failures, mobility, workload changes, etc.) while utilising limited resources. Moreover, these characteristics support dynamic and fast migration and composition of microservices across distributed resources, thus improving deployment flexibility. Thus, MSA demonstrates the potential to utilise Fog environments and achieve a balance between federated Fog and Cloud usage to improve application performance and meet QoS requirements.

1.2 Microservices-based IoT Application Placement in Fog Computing Environments

1.2.1 Problem Definition and Challenges. Microservices-based IoT Application Placement within Fog computing environments falls under **Fog Application Placement Problem (FAPP)** addressed in works such as References [12, 43, 90, 94]. FAPP addresses application deployment and maintenance within Fog environments, where services with agreed **Service Level Agreements (SLA)** are deployed onto federated Fog and Cloud resources for the shared use by the application users. To this end, FAPP considers factors such as horizontal and vertical scalability, request load balancing, ubiquitous access, location awareness, and fault tolerance to produce application deployments that meet the required performance of the application services [12, 43].

Based on the definition of FAPP, we define "microservices-based IoT application placement in Fog environments" as follows:

Let A be a microservices-based IoT application where A consists of a set of independently deployable and scalable microservices (M_A) and a set of services provided to the application users (S_A). We use the term "service" to denote end-user-requested business functionalities, which can be either an atomic service (consisting of only a single microservice) or composite services (composed of multiple microservices). As the microservices are granular with well-defined business boundaries,
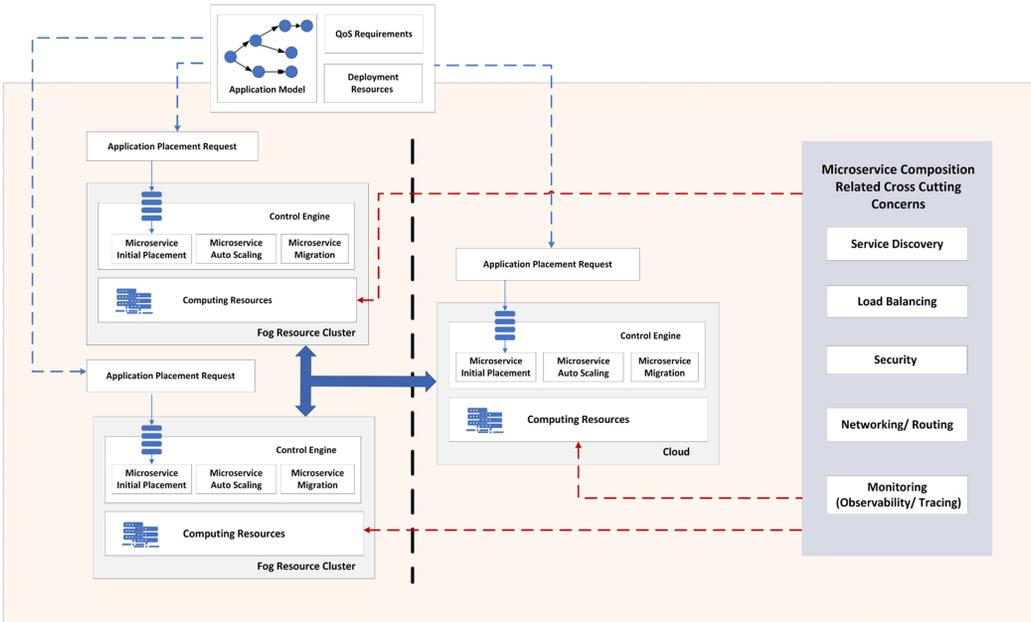
Fig. 2. Microservices-based fog application placement problem.

they ($m \in M_A$) can communicate using lightweight protocols to create composite services ($s \in S_A$) requested by the end-users. Placement of such applications includes mapping these microservices to distributed Fog and Cloud resources such that the requirements (i.e., resource requirements of the microservices, QoS requirements of the services, etc.) are ensured while maintaining seamless connectivity across distributed microservices to create composite services. Figure 2 provides a motivation scenario to elaborate this further. The event flow of the Fog application placement process shown in Figure 2 includes distributed control engines placed within Fog and Cloud environments receiving application placement requests from application providers and processing them for placement across Fog device clusters and Cloud data centres using efficient and scalable placement algorithms. Control engines can process them sequentially or as a batch based on the defined placement policy. Placement policy uses knowledge about the application model, QoS requirements, and microservice composition-related features to produce a resultant placement for the microservices.

To emphasise the novel aspects of microservices-based application placement compared to other application architectures, we identify four main challenges related to solving the placement problem, namely, challenges related to application modelling, developing microservices placement policy, microservice composition, and performance evaluation of the placement policies. Figure 3 depicts the relationships among these challenges to emphasise the importance of their collective consideration to produce efficient placements for microservices-based IoT applications. We discuss each challenge in detail below to identify their essential aspects and construct a taxonomy (see Figure 4) based on the discussed challenges and characteristics.

(1) *Challenges related to accurate modelling of MSA*: The precise formulation of the placement problem depends on accurate modelling of the application architecture, such that all essential characteristics of the architecture are properly abstracted and captured to support the placement of an extensive range of IoT applications developed following the said architecture. When considering MSA, the fine-grained nature of the microservices and their complex interaction patterns
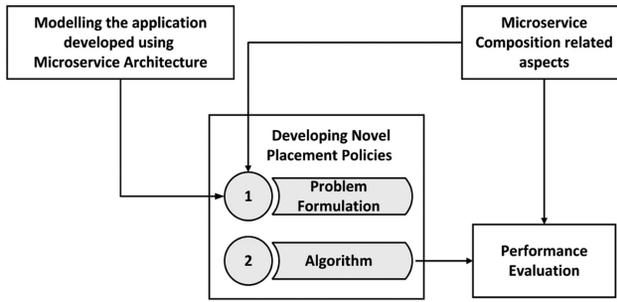
Fig. 3. Main challenges related to designing novel Fog placement policies for microservice-based IoT applications and their relationships.
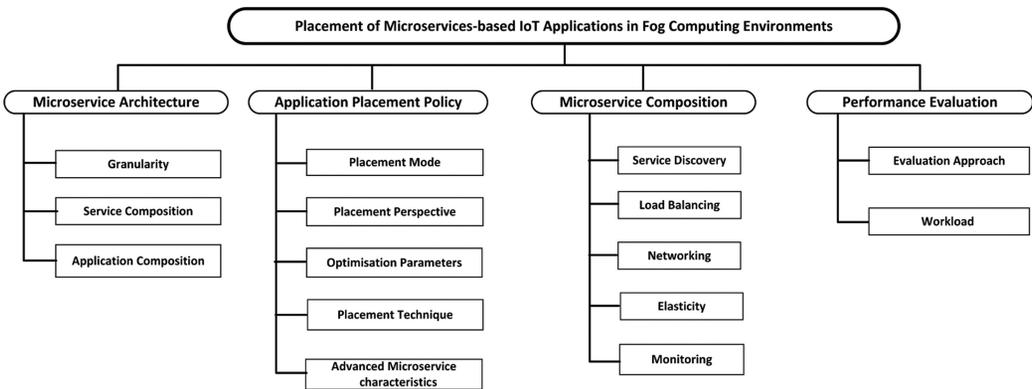
Fig. 4. Taxonomy for microservices-based IoT applications placement.
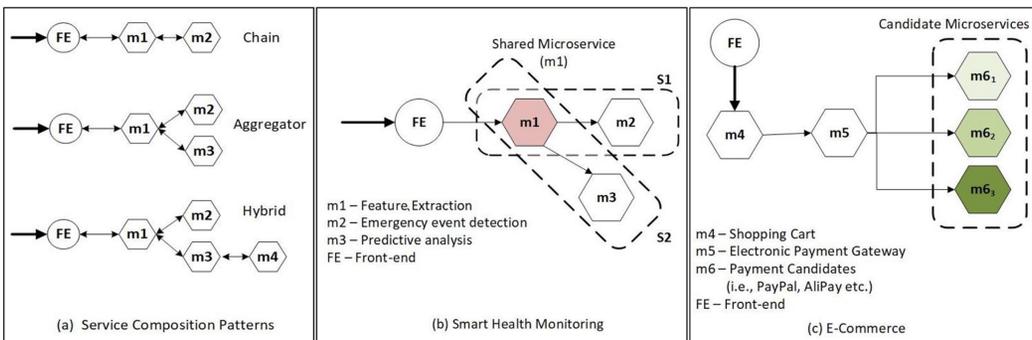
Fig. 5. Example use cases of microservice composition.

set MSA apart from other application architectures and add novel challenges to the placement problem formulation.

We explain these characteristics through a couple of case studies (see Figure 5). As depicted in Figure 5(b), granular and inter-operable microservices enable applications to be developed as a collection of composite services with heterogeneous QoS requirements. Here, the Smart healthcare application for patient monitoring has two services: emergency notification service for detecting

abnormalities in real-time (service S1) and latency-tolerant service for AI-based predictive analysis to give early health warnings (service S2) [80]. Moreover, such granular decomposition of an application into microservices following the separation of concern principle results in heterogeneous microservices regarding resource requirements (e.g., m1 and m2 with CPU requirements and m3 with GPU requirements). Furthermore, being granular and loosely coupled components, microservices create complex interaction patterns such as shared microservices among composite services (e.g., m1 is part of both S1 and S2), candidature microservices creating conditional dataflows based on the user demand, and third-party microservices (e.g., m6 in Figure 5(c) depicting a payment gateway with multiple optional microservices for request routing) that add more constraints on latency, security, scalability, and so on, during placement. Depending on the interactions among microservices, the dataflows and possible data paths can vary, too. Thus, unlike in the previous works that solved the placement and scheduling of workflows with directional acyclic dataflows [87], MSA can have composite services with bi-direction and cyclic dataflows where one microservice invokes other microservices, receives responses from them, and finally aggregate and further process the responses before returning the results of the composite service (see Figure 5(a)) [53, 80].

Placement policies can use the knowledge of such characteristics to utilise the distributed resources across multiple Fog clusters and Cloud data centres to overcome the challenge of limited Fog resources and meet heterogeneous resource requirements and QoS requirements while ensuring seamless connectivity among interacting microservices (e.g., placing m1 and m2 in the Fog layer closer to the network edge to meet stringent latency requirements [79] while placing m3 in the Cloud to meet the resource requirements, determining the number of horizontally scaled microservice instances to be placed for $m6_1$, $m6_2$, $m6_3$ based on their demand [114], calculation of latencies depending on different dataflows within composite services [80], etc.)

Thus, to utilise MSA's full potential and overcome related challenges, accurate modelling of the applications is of paramount importance. This includes the correct depiction of microservice granularity (i.e., number of microservices, microservice heterogeneity and their invocation patterns), service composition (i.e., number of microservices per service and their dataflow patterns), and application composition (i.e., number of heterogeneous services per application and use of shared/candidature/third-party microservices).

Challenges related to developing microservices placement policy: Developing a novel placement policy includes problem formulation, which contains optimisation metrics, objectives, constraints, and so on, and, based on that, creating an efficient algorithm to reach an optimum placement. Compared to other application models, the loosely coupled nature and the resultant complex interaction patterns, along with higher dynamism, affects different aspects of the problem formulation, such as QoS granularity (i.e., latency requirements can be defined per composite service or among interacting microservices), QoS-awareness (i.e., handling competing QoS requirements among multiple composite services with shared microservices in a QoS-aware manner as in Figure 5(b)), incorporating the horizontally and vertically scaled placement of microservices across federated Fog, and Cloud environments and load balancing requests among dynamically composed distributed instances [26, 80, 114] to utilise limited resources better. This provides opportunities to create efficient redundant placements, optimum request routing-based placements, locality-aware placements, resource contention-aware batch placements, and so on, to meet the expected performance of the services.

While these aspects increase the complexity of the placement process, their incorporation in placement policies contributes to optimum and balanced utilisation of distributed, heterogeneous, and resource-constrained Fog devices and resource-rich, centralised Cloud resources to improve the performance of heterogeneous IoT application services.

Challenges related to microservice composition: The composition of granular microservices to create composite services becomes a critical challenge affecting the application's performance. The main challenges related to this include cross-cutting functions such as service discovery, load balancing, monitoring, networking and so on, that come with distributed placement, scalability, elasticity, migration, redundant deployment, and failures of microservices. Furthermore, as a cloud-native application architecture that can be extended to the distributed Fog layer, these composition-related functions need to support higher flexibility and dynamism across distributed environments, which separates microservice composition from previous web services-oriented SOA. Thus, container orchestration or choreography frameworks and service mesh technologies are introduced to handle the dynamic changes and maintain interconnections among microservices minimising adverse effects on service performance.

Moreover, knowledge about their capabilities is essential in developing placement policies and, in turn, is vital in creating evaluation platforms (simulators and real-world test beds) to evaluate the performance of the developed policies. To this end, factors such as load-balancing policies, overheads due to dynamic service discovery, ability to have service discovery, and load-balancing across multiple computing environments needs to be incorporated into placement problem formulation [50, 52, 108] to achieve efficient placements. Moreover, the evaluation of placement policies should be carried out with the accurate implementation of these cross-cutting concerns to properly evaluate the placement policies.

Challenges related to performance evaluation: Accurate evaluation of placement policies depends on the used Fog framework/platform and the workloads. Due to the lack of commercial Fog service providers providing Fog platforms such as IaaS, PaaS, or SaaS, the evaluation of the novel placement approaches is mainly handled by simulations or small-scale Fog computing frameworks developed by researchers. Furthermore, with microservices architecture, simulators and frameworks need to be extended further with container orchestration/choreography support, MSA-related cross-cutting function support through service mesh technologies, distributed monitoring, dynamic placement policy integration, and so on. Moreover, the workloads used for evaluations should adequately capture the complex characteristics of the MSA and large-scale IoT applications to achieve accurate evaluations.

1.2.2 Motivation for Research. MSA is becoming exceedingly popular for the design and development of large-scale IoT applications, and the features of the architecture (loosely coupled nature, fine granularity, extensibility, cohesiveness, scalability, etc.) demonstrate immense potential to improve the performance of the applications through their efficient placement within federated multi-Fog multi-Cloud environments. As a cloud-native application architecture, MSA supports dynamic composition of loosely coupled microservice that can be scaled up/down across geo-distributed Fog environments and distributed across Fog resources and Cloud data centres to meet the throughput demand of the applications in a QoS-aware manner. Thus, designing efficient and scalable algorithms for the "Placement of Microservices-based IoT Applications in Fog Computing Environments" is a vital research area. As explained in the section above, to develop efficient placement algorithms, it is paramount to incorporate the characteristics, features, and challenges of the MSA into the placement problem formulation and also create evaluation platforms accordingly. Even though the research on microservices-based application placement is still in its early stages, due to the rapid adaptation of microservices in IoT application development, it is essential to analyse existing works that focus on microservices placement in Fog environments to create a comprehensive picture of the current status of research and what research gaps remain to be addressed in the future research.

To this end, we created a taxonomy based on the characteristics and challenges discussed in previous sections (depicted in Figure 4). In the following sections, we discuss each aspect, propose

separate low-level taxonomies for them, analyse existing literature under each taxonomy and identify areas of improvement. To create the taxonomy, we have carefully selected 45 research papers published in 2018–2022, with 78% of the selected papers from 2020–2022. All the selected papers comprehensively explore above discussed aspects related to the placement of microservices-based IoT applications in Fog computing environments. This would provide a holistic view of multiple aspects related to problem formulation of microservices-based application placement, including; how existing works model MSA-related characteristics, how QoS parameters and constraints are modelled with respect to application architecture, what objectives are handled by the placement algorithms and how microservice characteristics facilitate them, details on microservice composition related cross-cutting functions and how they improve the dynamic nature of the microservices thus enabling better placement policies, and finally the status of evaluation platforms and workloads that can be used to evaluate placement policies accurately. Hence, this survey aims to act as a knowledge base for future research to comprehensively address the challenges of the microservices-based IoT applications placement in Fog computing environments while reaping the benefits of the architecture. The main contributions of our work are as follows:

- We present a comprehensive background on MSA, IoT applications, Fog computing, and their integration to identify unique characteristics and challenges that differentiate "Microservices-based Application Placement in Fog environments" from other application models.
- We review recent research on microservices-based application placement in Fog environments and analyse them under multiple aspects related to the placement problem, including modelling microservices-based applications, creating application placement policies, microservice composition, and performance evaluation and propose taxonomies for each of the above perspectives. Our taxonomies capture MSA-related features and challenges in application placement.
- We identify research gaps in microservices-based application placement in Fog environments with reference to each proposed taxonomy.
- We identify and propose future research directions that researchers can use to improve the Fog application placement further.

1.3 Article Organization

The rest of the article is organised as follows: Section 2 presents a qualitative analysis of existing surveys and compares them with our work. The proceeding sections introduce and discuss detailed taxonomies under each aspect identified in our high-level taxonomy: Section 3 introduces the taxonomy on modelling IoT applications developed according to MSA and analyses existing works, Section 4 presents the taxonomy for placement policy design for microservices-based applications and discusses the current works accordingly, Section 5 introduces the taxonomy for microservice composition, and Section 6 presents the taxonomy for performance evaluation. Based on the gaps identified in each section, Section 7 provides future research directions. Finally, Section 8 summarises this survey.

2 RELATED SURVEYS

We analyse related surveys belonging to three main areas of research; surveys on Edge and Fog computing, Osmotic computing, and MSA, and conduct a qualitative comparison of the aspects covered by each of these surveys to define our contribution. Table 1 presents a comparison of the features covered in the surveys.

Existing surveys on Edge and Fog computing cover a wide range of research areas, such as resource management, application placement, and application/task scheduling. Surveys on Fog

Table 1. Summary of Existing Surveys

Work	MSA	Research Areas				Discussion on Application Model			MSA-specific discussions				Placement Evaluation	
		Edge/Fog Computing			Osmotic Computing	Available	Depth		Modelling Application	Microservice Composition	Relates application model to placement problem formulation	Relates microservice composition to placement problem formulation	Fog Platforms	Microservice Workloads
		Resource Management	Application Placement	Application/ Task Scheduling			High-Level Taxonomy	Architecture Specific Taxonomy						
[51]	O	✓	Δ	O	O	O	O	O	O	O	O	O	O	O
[55]	O	✓	O	✓	O	Δ	Δ	O	O	O	O	O	O	O
[53]	Δ	O	✓	O	O	✓	✓	O	O	O	O	O	O	O
[70]	O	Δ	✓	O	O	✓	✓	O	O	O	O	O	O	O
[95]	O	O	✓	O	O	✓	✓	O	Δ	O	Δ	O	Δ	O
[60]	✓	O	✓	O	O	Δ	O	O	Δ	Δ	O	O	Δ	O
[71]	O	O	✓	O	O	Δ	O	O	O	O	O	O	Δ	O
[74]	O	O	✓	O	O	Δ	O	O	O	O	O	O	O	O
[41]	O	O	Δ	✓	O	✓	✓	O	O	O	O	O	Δ	O
[12]	O	O	✓	O	O	Δ	O	O	Δ	O	Δ	O	O	O
[90]	O	O	✓	O	O	✓	✓	O	O	O	O	O	Δ	O
[99]	O	Δ	Δ	Δ	O	✓	✓	O	O	O	O	O	O	O
[57]	✓	O	Δ	O	Δ	✓	O	Δ	Δ	Δ	O	O	O	O
[84]	✓	O	O	O	O	✓	O	O	Δ	Δ	O	O	O	O
[39]	✓	O	O	O	O	✓	O	Δ	Δ	Δ	O	O	O	O
[75]	✓	O	O	O	✓	Δ	O	O	O	Δ	O	O	O	O
[6]	✓	O	O	O	✓	O	O	O	O	O	O	O	O	O
Our	✓	O	✓	O	Δ	✓	O	✓	✓	✓	✓	✓	✓	✓

✓: Discussed, Δ: Partially Discussed, O: Not Discussed.

resource management mainly focus on the architecture and characteristics of underlying Fog infrastructure (i.e., virtualisation, tenancy, etc.) and algorithms used in administrative operations, including device discovery, monitoring, performance benchmarking, load distribution, autoscaling, and so on. Fog application placement-related surveys comprehensively analyse the broad range of research work that maps applications to Fog resources to meet their non-functional requirements. In contrast, surveys on Fog application/task scheduling discuss distributing and sequencing ephemeral tasks (i.e., independent tasks or dependent tasks of the application arranged as workflows) for execution within Fog resources. These surveys also discuss works related to the computation-offloading problem in Edge and Fog computing. Thus, scheduling problem considers tasks or workflows with ephemeral life cycles that are deployed for use by a particular user, whereas Fog application placement considers deploying and managing applications with perpetual life cycles with application services accessed by a large number of users.

Hong et al. [51] focus on resource management and, thus, study and classify the architecture of Edge and Fog platforms and algorithms designed for resource management. Their discussions on application placement are limited to analysing how static and dynamic characteristics of the resources are incorporated into placement algorithms for resource allocation. Hence, they do not focus on application architecture-related characteristics and identifying their effect on efficient placement. Jamil et al. [55] propose a taxonomy of optimisation metrics and algorithms used in resource allocation and task scheduling in Fog computing. However, they do not characterise existing works based on the application models or architectures and fail to analyse optimisation characteristics with respect to them. Brogi et al. [12], Salaht et al. [90], Mahmud et al. [70], Smolka et al. [95], Kaur et al. [60], Malazi et al. [71], Nayeri et al. [74], and Islam et al. [53] discuss **Fog Application Placement Problem (FAPP)**. In Brogi et al. [12], Kaur et al. [60], Malazi et al. [71], and Nayeri et al. [74], the main focus is on analysing placement algorithms based on the methodologies of solving the placement problem, their constraints, and optimisation metrics. Their discussions on application models are limited to modelling application module dependencies as constraints of the placement problem formulation. Salaht et al. [90], Mahmud et al. [70], and Smolka et al. [95] provide a high-level taxonomy on covering the breadth of Fog application placement. Classifications of the application models in References [90] and [70] are limited to high-level taxonomies that categorise existing works under monolithic architecture and distributed architectures such

as modular and microservices. Reference [95] also provides a high-level taxonomy for application properties highlighting structure and parameters used in modelling the application, but does not focus specifically on microservices-related characteristics. Islam et al. [53] also provide a high-level taxonomy that classifies applications into monoliths and distributed architecture and limits the discussion on MSA to providing future research directions for microservices-based application placement. Thus, the above works do not capture MSA-related characteristics and challenges related to solving the placement problem of microservices-based IoT applications.

Goudarzi et al. [41] also focus on providing a high-level taxonomy to give a comprehensive and broad overview of IoT application scheduling in Fog environments where the authors discuss general aspects related to scheduling, such as application structure, environmental architecture, optimisation characteristics, decision engine characteristics, and performance evaluation of algorithms. As their survey focuses on providing a high-level taxonomy of each aspect, the taxonomy of the application architecture is limited to a high-level categorisation of applications (monolithic, independent, loosely coupled, etc.). This work introduces microservices architecture as an example of loosely coupled application structure but does not carry out further discussion on the characteristics of the architecture. An in-depth analysis of MSA is out of the scope for this survey, microservice composition-related cross-cutting functionalities (i.e., load-balancing, service-discovery, networking, distributed monitoring and tracing, etc.), tools enabling dynamic composition (i.e., container orchestrators, service meshes, etc.), incorporation of MSA-related features into problem formulation and algorithm evaluation (i.e., Fog computing platforms that support dynamic microservice composition, microservice workloads) are not discussed in their work. Moreover, as the main focus is on scheduling problem, FAPP-related aspects such as throughput-aware horizontal scaling, redundant placements, load-balancing, location-awareness to support distributed users, and so on, are not discussed in their work.

MSA-related surveys mainly focus on development, operational phase concerns [39, 57], and challenges related to adaptation of MSA for application development [84]. Joseph et al. [57] provide a broad taxonomy using research work that captures development aspects related to MSA, such as modelling, architectural patterns, maintenance, testing and quality assurance, along with operational aspects, including placement, migration, service discovery, and load balancing. Although Reference [57] discusses Osmotic computing and Edge/Fog computing as distributed computing paradigms employing MSA, a categorisation of existing works within these paradigms is not performed. Reference [39] also provides a taxonomy covering all aspects of the microservices lifecycle. Taxonomy provided in Reference [39] is helpful for application developers adapting MSA to design and implement their applications, whereas our survey focuses on large-scale placement of multiple diverse microservices-based applications within distributed Fog environments. Thus, in contrast to Reference [39], we explore how to incorporate microservice characteristics into the Fog application placement problem to generate optimum placements. Razzaq et al. [84] study existing research to identify MSA-related software architectural styles, patterns, models, and reference architectures adapted by IoT systems. However, the placement aspects of such applications are out of the scope of their study. Thus, microservices-related surveys mainly focus on the design, development, and maintenance aspects of MSA in general without focusing on challenges related to their placement and deployment within distributed computing paradigms such as Edge and Fog computing.

Osmotic computing-related surveys focus on detailing the concepts and features of the Osmotic computing paradigm along with challenges and future directions [58, 101]. Androcec et al. [6] and Neha et al. [75] conduct a systematic review to capture the current status of Osmotic computing by analysing applications that follow the Osmotic computing principles, Osmotic computing-related topics addressed, and their level of maturity. While these works highlight the concepts and
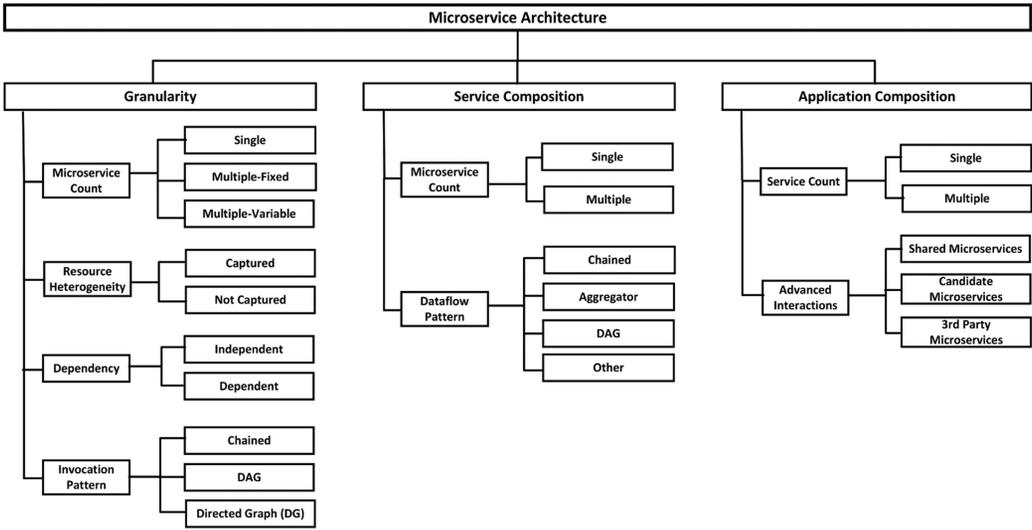
Fig. 6. Taxonomy for modelling of Microservice Architecture for placement problem formulation.

potential of Osmotic computing, they do not analyse existing works based on placement-related aspects, including features and challenges related to modelling, placement, service composition, and evaluation with respect to MSA.

Table 1 compares existing surveys with our work by presenting a summary of the key aspects covered. Developing efficient placement approaches for microservices-based applications requires modelling applications to capture MSA-related characteristics, placement policy creation by incorporating the application model-specific features and microservice composition-related characteristics to the placement problem formulation, and proper evaluation of the generated policies using Fog platforms that support microservice composition and microservices-based workloads. Based on the qualitative comparison, existing surveys fail to provide a thorough taxonomy to capture the above. In this work, we propose taxonomies for modelling applications based on MSA, placement policy creation, microservice composition, and performance evaluation and discuss their relationships. Moreover, we identify research gaps with respect to each aspect and propose future research directions.

3 MICROSERVICE ARCHITECTURE

To utilise the capabilities of MSA and overcome the challenges of the architecture within Fog environments, proper modelling of the applications is of vital importance so the placement algorithms can capture all aspects of the placement problem. To this end, Figure 6 presents the taxonomy for modelling applications based on MSA where we analyse microservices-based application modelling at multiple levels: granularity at microservice level (*Granularity*), the composition of microservices at the service level (*Service Composition*), and composition of services at the application level (*Application Composition*). Table 2 maps the existing works to the proposed taxonomy based on how each research work models the microservices-based applications.

3.1 Granularity

Granularity is one of the most important and challenging aspects of MSA. The fine-grained nature of the microservices allows application services to be depicted as a collection of small components communicating together to perform a certain end-user-requested service. While this allows

Table 2. Analysis of Existing Literature Based on the Taxonomy for Modelling of Microservice Architecture Presented in Figure 6

Work	Granularity				Service Composition		Application Composition			
	Microservice count	Resource Heterogeneity	Dependency	Invocation pattern	Microservice count	Dataflow pattern	Service count	Advanced Interactions		
								Shared	Candidate	3rd Party
[32]	Multi-V	Captured (P_C)	Dependent	Ch	Multiple	Ch	NA	-	-	-
[62]	Multi-V	Captured (RU)	Dependent	DG	Multiple	ND	Single	-	-	-
[44]	Multi-V	Captured (P_C)	Dependent	DG	Multiple	NA	Multiple	-	-	-
[29]	Multi-F	Captured (R,P_C,S)	Dependent	Ch	Multiple	Ch	Single	-	-	-
[79]	Multi-V	Captured (R,P_C,S,B)	Dependent	DAG	Multiple	NA	Multiple	✓	-	-
[43]	Multi-V	Captured (RU)	Dependent	DAG	NA	NA	NA	-	-	-
[30]	Multi-V	Captured (R,P_C,S,B)	Dependent	DAG	Multiple	DAG	Single	-	-	-
[103]	Single	NC	Independent	NA	Single	NA	Single	-	-	-
[81]	Mult-V	Captured (P_C, R)	Dependent	ND	ND	ND	ND	-	-	-
[91]	ND	Captured (RU)	Independent	NA	ND	NA	ND	-	-	-
[28]	Multi-V	Captured (P_C)	Dependent	ND	Multiple	Ch	Multiple	✓	-	-
[1]	Single	Captured (P_C)	Independent	NA	Single	NA	Single	-	-	-
[31]	Multi-V	Captured(R,P_C,P_G,S)	Dependent	DAG	Multiple	DAG	ND	-	-	-
[113]	Multi-V	Captured (P_C)	Dependent	Ch	Multiple	Ch	Single	-	✓	-
[26]	Multi-V	Captured (R,P_C,S)	Dependent	Ch	Multiple	Ch	Single	-	-	-
[108]	Multi-V	NC	Dependent	Ch	Multiple	Ch	ND	-	-	-
[52]	Multi-V	Captured (P_C)	Dependent	Ch	Multiple	Ch	ND	-	-	-
[61]	Multi-V	Captured (P_C)	Dependent	DAG	Multiple	DAG	Multiple	-	-	
[7]	Multi-V	Captured (P_C,R,S)	Dependent	DG	Multiple	DG	ND	-	-	-
[109]	Single	NC	Independent	NA	Single	NA	Single	-	-	-
[8]	Single	NC	Independent	NA	Single	NA	ND	-	-	-
[50]	Multi-V	Captured (R)	Dependent	Ch	Multiple	Ch	Multiple	✓	-	-
[48]	Multi-V	Captured (R,P_C,S)	Dependent	DAG	Multiple	Ch, Ag, H	Multiple	✓	-	-
[105]	Multi-V	Captured (P_C, R)	Dependent	DG	Multiple	DG	Single	-	✓	✓
[36]	Multi-V	Captured (P_C, R, B)	Dependent	DAG	Multiple	DAG	ND	-	-	-
[4]	Single	Captured (R,B)	Independent	NA	Single	NA	ND	-	-	-
[35]	Multi-V	Captured (P_C, R, B)	Dependent	DAG	Multiple	DAG	ND	-	-	-
[80]	Multi-V	Captured (R,P_C,S)	Dependent	DAG	Multiple	Ch, Ag, H	Multiple	✓	-	-
[45]	Multi-V	NC	Dependent	DAG	Multiple	DAG	ND	-	-	-
[67]	Multi-V	Captured (P_C,R)	Dependent	NA	Multiple	UWG	NA	-	-	-
[112]	Multi-V	Captured (P_G)	Dependent	DAG	Multiple	DAG	Single	-	-	-
[59]	Multi-V	Captured (RU)	Dependent	NA	Multiple	DG	ND	-	-	-
[40]	Multi-V	Captured (R,P_C,S,B)	Independent	NA	Single	NA	Multiple	-	-	-
[96]	Multi-V	Captured (P_C,S)	Dependent	Ch	Multiple	Ch	Single	-	-	-
[64]	Multi-V	NC	Dependent	Ch	Multiple	Ch	Single	-	-	-
[11]	Single	Captured (R,P_C)	Independent	NA	Single	NA	Single	-	-	-
[2]	Single	Captured (R,P_C)	Independent	NA	ND	ND	ND	-	-	-

Mult-F:Multiple-Fixed, Mult-V:Multiple-Variable, Ch:Chain, Ag:Aggregator, H:Hybrid, DAG:Directed Acyclic Graph, DG:Directed Graph, UWG: Undirected Weighted Graph, R:RAM, P_C:Processing power (CPU), P_G:Processing power (GPU), S:Storage, B:Bandwidth, RU:Resource Units, NC:Not Captured, ND: Not Defined, NA:Not Applicable.

QoS-improved placement within resource-constrained Fog devices and dynamic movement between federated Fog-Cloud environments following the Osmotic computing paradigm, it introduces complexities in interaction patterns among the microservices. Thus, when modelling the application placement problem, the level of granularity should be captured accurately to overcome the challenges while utilising the advantages introduced by the granular design.

Hence, we analyse the microservice granularity within IoT applications as follows:

(1) *Microservice count*: MSA decomposes the application into a set of microservices, increasing the complexity of the placement problem as the number of microservices increases. Thus, existing works capture different levels of granularity based on the number of microservices in the modelled IoT applications. Each application modelled in References [4, 8, 103, 109] consists of a single microservice such that the microservice is designed to perform a specific task requested by the end-user. As an example scenario, Reference [103] introduces an object detection application used by autonomous cars for the detection of other vehicles, pedestrians, road signs and so on, that consists of a single microservice for object detection service. To avoid the complexities introduced by having a large number of interconnected microservices, Reference [29] simplifies the placement problem by designing the placement algorithm to handle applications with a fixed number of microservices. Reference [29] proposes the

placement algorithm for applications consisting of two microservices: a high-throughput microservice that receives data and pre-processes it to reduce the throughput, and a low-throughput microservice that processes the data sent from the first microservice. Works such as References [30, 43, 62, 80, 113] remove this constraint and model the application as a collection of any number of microservices, thus providing robust placement algorithms that capture the problem-domain dependent granularity levels of MSA more accurately.

(2) *Resource Heterogeneity*: One of the advantages of decomposing applications into microservices is to achieve functional separation where the application is divided into separate modules following the "separation of concern" design pattern. This results in the separation of microservices based on their resource requirements as well (e.g., CPU, GPU, RAM, storage etc.). It is especially advantageous in Edge/Fog environments where resources are heterogeneous (e.g., Raspberry Pi,[1] Jetson Nano,[2] Dell PowerEdge XR12,[3] Lenovo ThinkEdge SE50,[4] etc.) and resource-constrained unlike in Cloud environments. References [29, 48, 80] demonstrate this by modelling microservices within the same application to have heterogeneous resource requirements in terms of multiple resource parameters such as RAM, CPU, storage, and bandwidth. References [31, 112] extend this to include GPU as well, where microservices with GPU requirements may have to be moved to different Fog locations or Fog service providers based on the GPU availability. While the above works represent resource requirements as a vector, some works such as References [43, 62] simplify the representation by introducing the scalar parameter "Resource Units" with the possibility of extending it to include multiple resource types.

(3) *Dependency among microservices*: Microservices are developed as independently deployable units with well-defined business boundaries such that their functionality is exposed to the outside through open interfaces. This allows microservices within applications to communicate easily with each other to create composite services. References [1, 91, 103] represent microservices as independent entities without any interconnections among them. In contrast, works such as References [32, 62, 79] model them to have dependencies, where microservices communicate with each other through lightweight communication protocols such as REST APIs and message brokers, creating a plethora of IoT services.

(4) *Invocation pattern*: The higher level of openness supported by microservices results in different invocation patterns among them, where client microservices invoke other microservices to perform functions required to complete the composite services. References [26, 32, 113] model the invocation relationship as a chained pattern. References [45, 48, 80] use a more general representation of microservice invocation by modelling it using DAG representation. References [44] and [62] model the invocation as directed graphs where the interactions between microservice are depicted using many-to-many consumption relationships.

3.2 Service Composition

With granularity comes the concept of composite services, where microservices interact to create services that perform a certain task and provide an output to the user. In this section, we analyse and categorise aspects related to service composition.

(1) *Microservice count*: The granularity of the microservices creates end-user services with varying numbers of microservices: "atomic services" consisting of a single microservice and

---

[1]https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/.
[2]https://developer.nvidia.com/embedded/jetson-nano.
[3]https://www.dell.com/en-au/work/shop/cty/pdp/spd/poweredge-xr12/aspexr12_vi_vp.
[4]https://psref.lenovo.com/syspool/Sys/PDF/ThinkEdge/ThinkEdge_SE50/ThinkEdge_SE50_Spec.pdf.

"composite services" that consist of multiple interconnected microservices. References [1, 103] represent each service by a single microservice that receives requests from the end-user front-end, completes a task, and provides the result back to the user without interacting with any other microservices. Other works such as References [26, 45, 80] model services with multiple microservices that interact together to perform tasks. Reference [26] describes a smart city application with a smart policing service used by the police to identify suspects where the service is a composition of three microservices, and References [79, 80] model a smart healthcare application with an emergency notification service where multiple microservices interact to detect abnormalities in ECG data streams and raise emergency alarms in real-time.

(2) *Dataflow Pattern*: Due to different interaction patterns among microservices, the dataflow within composite services can take many forms. References [28, 29, 32] consider the chained composition of microservices, whereas References [48, 80] model the services considering chained, aggregator, and hybrid dataflow patterns to create composite services. Works such as References [31, 61] model dataflow among microservices as DAGs, assuming the absence of cyclic dataflows while References [7, 105] include cyclic dataflows and represent the dataflows using DGs. Reference [67] models the interaction pattern using an **Undirected Weighted Graph (UWG)** where edges are represented using interaction weights irrespective of the direction of communication.

3.3 Application Composition

As the capabilities of IoT applications improve rapidly, they quickly evolve into complex applications covering large business domains due to the flexibility of design and development using MSA. We analyse the aspects related to this as follows:

(1) *Service count*: With the increase in connected devices and the generation of diverse data, IoT applications have evolved to provide many services to users. Thus, microservices-based IoT applications are modelled as a composition of one or more services. References [29, 103] define applications with single services, whereas References [28, 44, 61] model applications with multiple end-user services with heterogeneous QoS requirements. References [79, 80] model a smart health care application with two primary services; emergency alarm generation service with stringent latency requirements and a latency tolerant long-term analysis service. Modelling the applications with multiple services allows the placement problem to capture competing QoS requirements within applications and paves the way to propose placement policies that can utilise Fog-Cloud resources more efficiently.

(2) *Advanced Interactions*: Due to the fine-grained nature of the microservices, along with well-defined functional boundaries and lightweight communication methods, advanced interactions among microservices are possible. Reference [80] models applications where some microservices (e.g., feature extraction, data cleaning, etc.) are part of multiple composite services that have heterogeneous latency requirements. Reference [113] models the services to have candidate microservices (e.g., online payment gateways) depending on the payment method used by the user. Thus, the application is modelled to have alternative data paths. Reference [4] handles the concept of having alternative paths by defining control structures for each composite service through conditional branching within some of the microservices. Another advanced capability of microservices is using third-party microservices through the use of APIs, which is modelled in Reference [4], where composite services are created by combining microservices developed by multiple developers and provided as services by multiple computing service providers.

3.4 Research Gaps

Based on the analysis of existing works presented in Table 2, we have identified the following gaps related to microservices-based application modelling:

(1) Existing works show shortcomings in several aspects in capturing the granularity of the MSA, such as lack of use in generalised invocation patterns (i.e., directed graphs) and capturing resource heterogeneity in different microservices within the same application (i.e., use of GPU, databases, etc.). Many works use a chained invocation of microservices without considering complex dependency patterns that can occur due to the openness of microservice design. For capturing resource heterogeneity, most works consider one or more parameters such as CPU, RAM, and storage. However, GPU or TPU requirements are scarcely considered. With the rise of EdgeAI workloads in IoT, such parameters and the constraints imposed by them play an important role in the placement logic.

(2) In service composition, the dataflow pattern is not adequately defined and utilised in the placement process of the majority of the works. Many works define chained dataflow or acyclic dataflows, disregarding cyclic dataflows, which affect the end-to-end latency calculations of the services.

(3) Under application composition, most works model each application to have a single composite service. Thus, heterogeneity in QoS requirements between different composite services within the same IoT application is not appropriately captured. This also hinders the application from having complex interaction patterns and dataflow patterns within the applications.

(4) In modelling the applications, most works do not consider complex interaction patterns of microservices, including shared microservices among different composite services, candidate microservices, and third-party microservices. The placement of shared microservices has to consider multiple, heterogeneous composite services that they are part of and place them so all non-functional requirements of the services are satisfied under resource contention. For efficient placement of the candidate services, knowledge of their demand and usage is vital. The use of third-party microservices in applications results in security, reliability, and performance challenges (i.e., latency, availability, transaction cost, etc.) that are out of the control of the application developers, which need to be accounted for during the placement phase to improve performance. Most of the works that consider these only analyse the effect of a single pattern, thus failing to capture the impact of multiple ones.

4 APPLICATION PLACEMENT POLICY

MSA introduces novel aspects (i.e., QoS granularity, scalability, lightweight/independent deployment, etc.) that can be utilised for better placement of the IoT applications while also giving rise to novel challenges (i.e., microservice dependencies, interactions, cascading failures, etc.). We consider these MSA-specific effects in addressing the application placement problem and create a novel taxonomy, as shown in Figure 7. Current works are mapped to the taxonomy to identify gaps and possible improvements (see Table 3).

4.1 Placement Mode

Placement mode represents the number of placement requests processed by the placement engine during each execution of the placement algorithm. Placement modes are categorised into two groups: sequential mode, where the placement algorithm queues the placement requests to process one after the other, and batch mode, where a set of applications are considered for placement simultaneously. In References [26, 79, 103], the placement engine uses a **First In First Out (FIFO)**
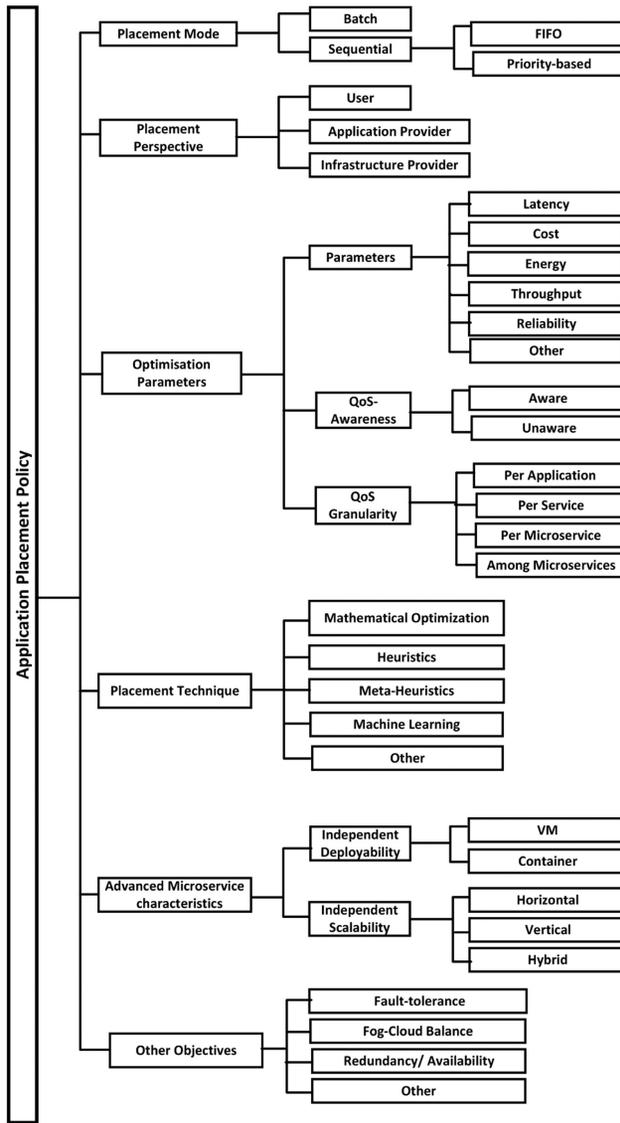
Fig. 7. Taxonomy for placement policies designed for microservices-based applications.

queue to store and process application placement requests sequentially. References [29, 30] prioritise the applications in the queue based on their resource requirements, whereas References [62] and [109] prioritise them based on the deadlines of the applications and order them for sequential placement. Placement engines in References [43, 61, 80] are designed using meta-heuristic algorithms (i.e., genetic algorithm, particle swarm optimisation) to handle the complexities of the batch placement and navigate the larger solution space successfully. Reference [31] employs a batch placement policy based on a heuristic algorithm to maximise the placement of IoT applications within a multi-domain federated Fog ecosystem under locality constraints imposed on microservices. Some works, such as References [1, 52, 108] do not properly define placement mode and carry out the evaluations using the placement of a single application.

Table 3. Analysis of Existing Literature Based on the Taxonomy for Application Placement Policy Presented in Figure 7

Work	Placement Mode	Placement Perspective	Placement Parameters		QoS Granularity	Placement Techniques	Adv /service characteristics		Other Objectives
			Parameters	QoS-awareness			Ind. Deployability	Ind. Scalability	
[32]	Seq-FIFO	UP	Latency	No	per microservice/s (Latency)	Heuristic – Greedy	VMs	NC	-
[62]	Seq-P	AP	Latency, Availability	Yes (Latency)	per application (Latency)	Heuristic	Containers	H	-
[44]	Seq-ND	UP	Latency	No	per service (Latency)	Heuristic	Containers	H	F-C Balance (D)
[29]	Seq-P	IP	Revenue, Latency Throughput	Yes (Latency, Throughput)	among microservices (Throughput) per application (Latency)	Heuristic-Greedy	Containers	NC	F-C Balance (S)
[43]	Batch	AP	Latency, RU_C	No	ND	Meta-Heuristic	Containers	H	Service Spread
[79]	Seq-FIFO	UP	Latency	Yes	per service (Latency)	Heuristic	Containers	H+V	F-C Balance (S)
[52]	ND	AP	Latency, Cost	No	per service	Meta-Heuristic	Containers	H+V	Redundancy
[81]	Seq-ND	AP	Cost, Reliability Latency	Yes (All)	per application (All)	Meta-Heuristic	ND	H	-
[108]	ND	AP	Latency, Energy	No	NC	Meta-Heuristic	Containers	H	Redundancy
[1]	ND	UP	Latency	Yes (Latency)	per microservice (Latency)	Machine Learning	Containers	H	Proactive Scaling
[91]	Batch	UP	Energy, Latency Cost	Yes (Latency)	per microservice (Latency)	Mathematical Programming (Lagrangian Multiplier)	Containers	H	Fault-tolerance
[61]	Batch	AP	Latency, Cost	Yes (Latency)	per service (Latency)	Meta-Heuristic	ND	NC	Fault-tolerance
[28]	Seq-ND	AP	Energy, Latency	No	NC	Meta-Heuristic	VMs	NC	-
[31]	Batch	IP	Revenue, Latency, Throughput	Yes (Latency, Throughput)	among microservices (Latency, Throughput)	Heuristic	Containers	NC	Locality-awareness
[30]	Seq-P	IP	Revenue, Latency Throughput	Yes (Throughput, Latency)	among microservice (Throughput) per application (Latency)	Heuristic-Greedy	Containers	NC	F-C Balance (D)
[103]	Seq-FIFO	AP/UP	Latency, Cost	Yes (Latency)	per application (Latency)	Reinforcement Learning	Containers	NC	Mobility-awareness
[7]	Seq-ND	IP	Latency, Throughput	Yes (All)	among microservices (All)	Meta-heuristic	Containers	H	F-C Balance (D)
[109]	Seq-P	UP+AP	Latency, Cost	Yes (Latency)	per microservice (Latency)	Heuristic – Greedy	Containers	NC	F-C Balance (D)
[48]	Seq-ND	AP	Latency, Cost	Yes (Cost)	among microservices (Latency) per application (Cost)	Heuristic – Greedy	Containers	H	-
[4]	Seq-ND	AP	Latency, RU_C Throughput	Yes (Throughput)	per microservice (Throughput, Latency)	Analytical Hierarchy Process (AHP)	Containers	NC	-
[36]	ND	IP+UP	Latency, Throughput Cost, RU_C, RU_N	Yes (Latency, Throughput)	per application (Latency, Throughput)	Approximate Algorithm + Deep Reinforcement Learning	Containers	H	Load-awareness Resource contention
[80]	Batch	AP	Latency, Cost Throughput, RU_C, RU_N	Yes (Latency, Throughput, Cost)	per service (Latency, Throughput, Cost)	Meta-Heuristic	Containers	H+V	F-C Balance (D)
[45]	Batch	UP	Latency, RU_N	Yes (Latency)	per service (Latency)	Meta-Heuristic	Containers	H	Location-awareness
[67]	ND	AP	Latency, RU_C	No	NC	Deep Reinforcement Learning + Heuristic	Containers	H	Load-awareness
[26]	Seq-FIFO	AP	Cost, Latency	Yes (Latency)	per service (All)	Mathematical Programming (Branch and Bound)	Containers	H+V	-
[114]	Seq-FIFO	AP	Latency	No	per service/application (Latency)	Monte carlo + Meta-heuristic	Containers	H	Availability
[50]	Batch	IP	Latency	No	per service (Latency)	Mathematical Programming (Gurobi MILP solver)	ND	H+V	Optimal routing
[105]	Seq-ND	UP	Latency, Throughput	Yes (Latency, Throughput)	among microservices (Latency, Throughput)	Mathematical Programming	VMs	H	Redundancy
[35]	ND	IP+UP	Latency, Throughput Cost, RU_C, RU_N	Yes (Latency, Throughput)	per application (Latency, Throughput)	Approximate Algorithm + Deep Reinforcement Learning	Containers	H	Load-awareness Resource contention
[112]	ND	UP	Latency, Throughput	Yes (Latency)	per service (Latency)	Heuristic-Greedy	ND	H	Resource contention
[72]	Batch	IP + UP	Latency, Energy Throughput	Yes (Latency, Throughput)	among microservices (Latency, Throughput)	Meta-heuristic	Containers	H	Fault-tolerance
[59]	Batch	UP	Latency	No	among microservices	Meta-Heuristic	Containers	NC	-
[40]	Batch	UP	Latency, Throughput, Energy	Yes (Latency)	per microservice	Meta-Heuristic	ND	NC	F-C Balance (S)
[96]	Seq-ND	AP	Cost, Latency	Yes (Latency)	per service (Latency)	Machine Learning	Containers	H + V	Location-awareness
[64]	Batch	UP+AP	Latency, RU_C	No	per service/application (Latency)	Meta-Heuristic	Containers	NC	-
[11]	Batch	AP	Latency, RU_C	Yes (Latency)	per microservice (Latency)	Meta-heuristic	ND	H	-
[2]	Batch	AP + IP	Throughput, RU_C	Yes (Throughput)	per microservice	Heuristic	Containers	NC	Resource Contention

FIFO:First-In-First-Out, P:Prioritised, UP:User Perspective, AP:Application provider Perspective, IP:Infrastructure provider Perspective, RU_C:Computation Resource Utilisation, RU_N:Network Resource Utilisation, ND:Not Defined, NC:Not Considered, H:Horizontal, V:Vertical, F-C Balance(D):Dynamic Fog-Cloud Balance, F-C Balance(S):Static Fog-Cloud Balance.

4.2 Placement Perspective

The placement perspective is identified based on the optimization parameters/objectives considered during the placement and from whose viewpoint they are addressed from. References [7, 29, 30] define the placement problem from the perspective of the Fog infrastructure provider, thus aiming to maximize the IaaS provider's revenue. Other works such as References [26, 28, 114] define the placement problem from the application provider perspective, where application providers expect the satisfaction of non-functional requirements (i.e., latency, throughput, reliability, etc.) of the services provided by their application while ensuring budget constraints related to Fog-Cloud deployment. References [32, 44] address this from a user perspective where the user sends a request for an application or service along with performance requirements, and the placement algorithm ensures the availability of the requested service under requested constraints (i.e., latency, throughput, etc.) to satisfy the user expectations.

4.3 Placement Parameters

In this section, we analyse the characteristics of the parameters considered by the placement policy.

(1) *Parameters:* IoT application services have multiple heterogeneous QoS parameters (i.e., latency [26, 50, 67, 105, 114], cost [26, 80, 109], throughput [4, 7, 105], energy [108], reliability [81], etc.) negotiated with the Edge/Fog infrastructure provider in the form of **Service Level Agreements (SLA)**. Moreover, placement decisions made from the infrastructure provider perspective consider the maximisation of the revenue of the IaaS provider. References [29] and [30] formulate the placement problem to maximise the revenue of the Fog provider, whereas References [7, 31] consider a federate Fog environment where each provider focuses on minimising the total deployment cost while minimising the number of resources rented from external Fog infrastructure providers and the Cloud. Resource utilisation is another parameter considered in current research where References [4, 43, 67] consider optimisation of computation resource usage, Reference [45] optimises network resource usage, while References [35, 80] consider both. Placement policies focus on optimising one or more of the above-mentioned QoS parameters. References [32, 44, 79] formulate the placement problem considering the satisfaction of a single parameter and focus on minimising the latency of the services deployed within Fog environments. References [61, 81, 91] focus on multiple parameters to reach a tradeoff between conflicting parameters, such as latency, cost, energy, and so on, by formulating the placement problem as a multi-objective optimisation problem.

(2) *QoS-awareness:* Reduction of service latency and core network congestion are two of the main objective of Edge/Fog computing paradigms. As a result, References [44, 52, 114] aim to place the microservices as close as possible to the user to reduce the overall latency of the application. While this is done in a QoS-unaware manner, assuming that all services require low latency, other research works such as References [31, 45, 80, 81] consider the QoS requirements of the IoT services before placing them. Such approaches highlight one of the main limitations of Edge/Fog computing, which is its resource-constrained nature compared to the Cloud data centres. Thus, they aim to capture the QoS heterogeneity of the IoT services and prioritise them based on their QoS requirements (i.e., stringent latency requirements over latency-tolerant services [31, 45], stringent budget constraints over higher budget availability [80, 81], throughput-aware placement/scaling of microservices [29, 80], etc.) and achieve a proper balance between Fog and Cloud resource usage in a QoS-aware manner. MSA further enhances this behaviour due to the ease of moving independently deployable microservices across Fog and Cloud data centres dynamically.

(3) *QoS Granularity:* Because of the fine-grained nature of the microservices and their possible composition patterns, QoS parameters are defined at different levels of granularity. Among existing works, QoS parameters are defined at three primary levels: microservice level, composite-service level, and application level. References [29, 30] define the throughput requirements at the microservice level, where it is presented as the bandwidth requirement between interacting microservices, whereas References [1, 32, 91] define latency requirements per each microservice. References [44, 52, 80] define latency, throughput, cost, and so on, per each composite service where microservices-based applications can consist of one or more such services. References [30, 103] define latency requirement per each application, assuming that the application performs a single function/service requested by the end-user. Most of the works define all QoS parameters at a single level, whereas References [29, 30] use different granularity levels based on the parameter (i.e., throughput at the microservice level and latency at the application level).

4.4 Placement Technique

Different placement techniques are chosen to implement the placement policies based on the complexity of the modelled microservices-based applications, placement mode, optimisation parameters and their granularity, and the dynamism of the considered scenario (in terms of distributed resources, workload, failures, etc.). References [26, 50, 91, 105] use mathematical programming to solve microservices-based application placement in Fog: Reference [50] formulates the problem using **Mixed Integer Linear Programming (MILP)** and solve it using Gurobi MILP solver,[5] Reference [91] uses the Lagrangian multipliers to solve the optimisation problem under multiple constraints including network QoS, price, resource usage, and so on. References [32, 44, 79] propose heuristic placement algorithms to optimise a single placement parameter such as latency. Heuristic approaches are proposed by works such as References [29, 31, 62] that consider multiple parameters as well. Here, References [29] and [31] formulate the placement problem as an Integer Linear Programming problem to solve it using greedy heuristic algorithms. References [43, 45, 61], which try to handle batch placement scenarios and multiple placement parameters, propose placement policies based on evolutionary meta-heuristic algorithms such as **Genetic Algorithm (GA), Particle Swarm Optimisation (PSO), Ant Colony Optimisation (ACO)** to navigate large solution space efficiently. With increased computation power available for placement engines, algorithms are moving towards **Machine Learning (ML)**: Reference [1] uses ML-based forecasting models to make predictive auto-scaling decisions, and References [35, 67, 103] use Reinforcement Learning-based approaches to tackle the highly dynamic nature of Edge/Fog environments and the microservices. Moreover, existing works use other techniques such as **Analytical Hierarchy Process (AHP)** [4], which is a powerful decision-analysis method used to prioritise and balance multiple criteria, and computation algorithms such as Monte Carlo method [114] used to capture uncertainties in the placement problem introduced by the MSA (i.e., candidate microservices, third-party microservices, etc.).

4.5 Advanced Microservice Characteristics

The popularity of the MSA for deployment within highly distributed Fog environments stems from two primary microservices-related characteristics, their independently deployable and scalable nature.

(1) *Independent Deployability:* Decomposing monolithic applications into a collection of microservices can improve deployment-related aspects such as fault tolerance and reliability

---

[5]https://www.gurobi.com/products/gurobi-optimizer/.

due to the independently deployable nature of the microservices. This is further supported by containerisation, which provides a lightweight method for deploying microservices compared to the earlier used **Virtual Machines (VMs)**. Reference [32] proposes a deployment scenario where all microservices of a user-requested service are mapped onto a single VM. The vast majority of existing works [31, 43, 62, 108] improve the deployment through the use of containers (i.e., Docker), which allows rapid deployment by providing microservices with lightweight, isolated environments. The use of containers enables fast deployment of microservices while mitigating single point of failure using the distributed placement of microservices. References [79, 91] highlight the importance of using container technology due to their faster spin-up times and analyse resultant performance improvements under dynamic conditions. References [26, 80] consider the cost of the containerised microservices by adapting the pricing models used in container platforms provided by the commercial Cloud providers (i.e., Amazon Fargate, Azure Containers).

(2) *Independent Scalability:* As microservices are independently deployable units with well-defined business boundaries, each microservice can be independently scaled to meet the throughput requirements. This is especially advantageous in Edge/Fog computing scenarios where devices are heterogeneous in their resource capacities; workloads also vary rapidly with the popularity of the services, mobility of the users, and so on. Scalability of the microservices is used in multiple ways where some works consider only horizontal scalability while others use a combination of horizontal and vertical scalability to satisfy throughput requirements. Reference [43] uses horizontal scalability of the microservices to spread instances of each microservice uniformly across the Fog landscape to support distributed access by users. References [79, 80] consider resource heterogeneity of the Fog devices and use a hybrid approach of vertical and horizontal scalability to meet the throughput requirement. Current works use the scalability of the microservices for multiple purposes. Above works focus on satisfying the overall throughput requirement while utilising resource-constrained Edge/Fog devices. Meanwhile, Reference [45] uses horizontal scalability to incorporate location awareness, whereas References [91, 114] use horizontal scalability to achieve redundancy, which improves availability and fault-tolerance of the application.

4.6 Other Placement Objectives

One of the main advantages of using MSA for Fog application development is the ability to easily utilise both Fog and Cloud resources to meet application requirements. This concept is also put forth in Osmotic computing, which highlights the importance of dynamically moving microservices between Cloud and Fog and achieving an equilibrium such that the non-functional requirements of the services are met. In References [29, 79], this is handled in a static manner where microservices to be placed on Cloud are predefined per each application based on the deadline requirement of the composite services each microservice belongs to. References [7, 30, 44, 80] handle this in a dynamic way: Reference [30] partitions the application based on the throughput requirement to place microservices with higher throughput requirements in the Fog and rest in the Cloud, Reference [44] analyses the popularity of the microservices based on the user requests to move less popular ones to the Cloud, Reference [7] achieves balance by considering the cost of deployment in Fog and Cloud, Reference [80] formulates a multi-objective problem to achieve a tradeoff between Fog device usage and network usage to achieve a balance between Fog and Cloud resource usage dynamically.

Locality or location awareness is another aspect that comes with IoT applications due to data security and latency requirements. Sensitive IoT data (e.g., healthcare, security camera footage, etc.) can have constraints on geographical locations for the processing, which requires certain

microservices to be placed within certain regions or within certain Fog service providers in federated Fog computing environments. Reference [31] adds locality constraint to their problem formulation and handles it using a heuristic approach. With the distributed nature of the Edge/Fog resources and the users, location awareness can be used to minimise network resource usage and delay [43, 45]. Reference [45] utilises this concept where the user's location is used to select the microservice instances such that the requests are routed to the closest instances. Reference [43] introduces a metric called service spread that evenly distributes microservices across the Fog landscape to improve performance when users are uniformly distributed.

Modularity and scalability of microservices pave the way for efficient redundant placement to improve the availability and fault-tolerance of the services. Reference [114] proposes a redundant placement policy for composite services consisting of chained microservices, considering the uncertainty of requests and heterogeneity of the resources. This work aims to reduce the outage time under failures through redundant placement of microservices. To address the challenge of cascading failures that occur in MSA, Reference [61] proposes a fault tolerance method for applications deployed using the API gateway pattern where the API gateway acts as the access point for requests coming from the users. This work deploys API gateways within cache-enabled edge nodes so data related to service requests can be cached proactively in case downstream microservices become unavailable due to failures until the microservice is redeployed using a reactive fault-tolerance approach.

Due to the horizontal scalability of microservices, proper load balancing and routing approaches are required to ensure performance requirements. To address this, References [50, 52, 108] formulate the application placement problem as a combination of microservice replica placement and determining the optimal dataflow path of composite services. Reference [52] uses minimisation of service latency and cost (both computation and data transfer costs) to achieve the optimum level of service replication and identification of the request path that minimises latency of the composite services. Reference [108] considers latency and power consumption as optimisation parameters to achieve this. Reference [50] extends this further by incorporating SDN controller placement as part of the problem, where SDN controllers are used for service discovery and determining dataflow.

Deploying multiple lightweight containers onto the same Fog device can result in resource contention, depending on the resource requirements of each container instance. To overcome this, References [35, 36, 112] propose online algorithms to detect shared-resource contentions and afterwards dynamically adjust resource allocations or migrate microservice instances to other idle nodes to maintain the required level of performance. These works capture different types of resource contentions, including I/O [35], GPU global memory bandwidth [112], and computation capacity [36] contentions.

4.7 Research Gaps

Based on the analysis of existing works presented in Table 3, we identified the following gaps related to microservices-based application placement:

(1) QoS-granularity and QoS-awareness, together with proper placement mode selection, can improve performance, especially when microservices-based IoT applications are considered. As IoT applications grow in complexity to provide many services with heterogeneous QoS requirements, the granularity of the microservices paves the way for per-service QoS definitions. Together with batch placement or sequential placement with QoS-aware prioritisation, this approach is rarely explored in existing works. The following shortcomings are apparent in existing works:
   - QoS-granularity: Most of the works define QoS requirements among interacting microservices (i.e., latency and throughput requirements between two microservices), which

becomes less feasible due to composite services with complex interaction patterns and their agile evolution as applications evolve. As a result, existing works scarcely capture the QoS heterogeneity of the composite services within the same application. Thus, MSA-related scenarios like competing requirements that exist due to shared microservices are not handled.

- QoS-awareness: Many works do not define QoS requirements and capture the heterogeneity but try to minimise overall latency, cost, and so on. This approach hinders the proper utilisation of limited Edge/Fog resources and limits achieving equilibrium between Fog-Cloud resource usage.

   The above gaps are tightly coupled with how the microservices-based application is modelled, and overcoming them requires the proper capturing of MSA as described in Section 3.

(2) In existing works, there is scope for incorporating advanced microservice characteristics such as the independently deployable and scalable nature of the microservices. Although many of the works use container technology for microservice deployment, they lack proper utilisation of the fast spin-up time, lightweight deployment capabilities and related cost models, whereas, for independent scalability, only a very few works explore the combination of horizontal and vertical scalability to support throughput requirements of the composite services. This can further consider the scalability constraints of different microservices (i.e., microservices with databases).

(3) Placement objectives of the current works lack emphasis on the following: security challenges related to microservice, utilising microservices for improvement of reliability and fault tolerance, dynamic microservice placement under federated Fog architectures, dynamic Fog-Cloud balance, mobility-aware placement, consideration for load uncertainty, and so on.

5 MICROSERVICE COMPOSITION

The complex interaction patterns among microservices, their ability to independently scale up/down to maintain performance and distributed deployment of microservice instances across networked devices is supported through microservice composition mechanisms. Figure 8 presents the taxonomy for essential aspects of the successful composition of microservices within distributed environments. For this analysis, we use both Edge/Fog frameworks designed for microservice deployment [33, 110] and conceptual frameworks/simulators used in formulating and solving placement problems [26, 48], as demonstrated in Table 4. We analyse the main functions related to microservice composition in the sections below.

5.1 Service Discovery

Dynamic placement algorithms proposed to handle microservice placement [67, 79, 110] define service discovery mechanisms so changes (i.e., scale up/down, failures) in microservice instances are made known to other microservices that interact with them. References [69, 79] use a client-side service discovery pattern where each client microservice queries a dynamically updated service registry to determine the available service instances. References [61, 67, 110] use server-side service discovery where the requests are directed to a designated entity (i.e., a load balancer, a proxy, etc.) that is responsible for directing the requests towards available service instances. "FogAtlas" [33], which is a Fog computing platform used by multiple works such as References [29, 31] and other works such as References [7, 110] use Kubernetes as the orchestrator along with its default proxy-based, server-side service discovery mechanisms. Reference [61] implements server-side service discovery using an API gateway as the ingress node responsible for service discovery and service composition.
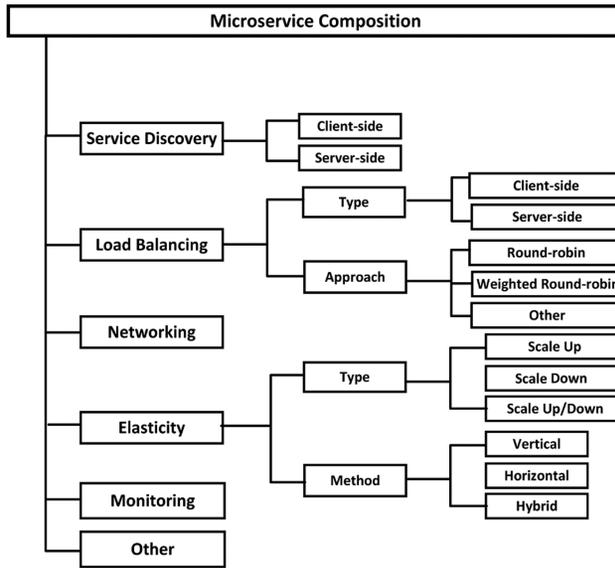
Fig. 8. Taxonomy for microservice composition.

Table 4. Analysis of Existing Literature Based on the Taxonomy
for Microservice Composition Proposed in Figure 8

Work	Service Discovery	Load Balancing		Networking	Elasticity		Monitoring	Other
		Type	Approach		Type	Method		
[33]	S-S	S-S	Round Robin	✓ (Kubernetes)	U-D	H	✓ (Prometheus)	-
[79]	C-S	C-S	Weighted Round Robin	-	U	H+V	-	-
[26]	NC	C-S	Round Robin	-	-	-	-	-
[80]	NC	C-S	Weighted Round Robin	-	-	-	-	-
[7]	S-S	S-S	Custom	✓ (Flannel + Istio)	-	-	✓ (Prometheus)	-
[67]	S-S	S-S	Custom	✓ (Kubernetes + Istio)	U-D	H	✓ (Prometheus)	-
[52]	ND	S-S	Custom	-	-	-	-	
[61]	S-S	NC	-	-	-	-	-	Fault-tolerance
[48]	ND	C-S	Round Robin	-	-	-	-	-
[110]	S-S	S-S	Weighted Round Robin	✓ (Kubernetes + KubeEdge)	-	-	✓ (Custom)	-
[69]	C-S	C-S	Weighted Round Robin	✓	U-D	H+V	-	-
[24]	S-S	S-S	ND	✓ (MQTT Broker)	U-D	H	✓ (Custom)	Fault-tolerance Security
[15]	S-S	S-S	ND	✓ (Kubernetes + CNI)	U-D	H	-	-
[96]	NC	ND	Weighted Round Robin	-	-	-	-	-

C-S: Client-side, S-S: Server-side, U: Up, D: Down, H: Horizontal, V: Vertical, CNI: Container Network Interface, ND: Not Defined, NC: Not Considered.

5.2   Load Balancing

(1) *Type:* Current works model load balancing using two primary approaches: *client-side load balancing* [26, 48, 79], where each client is responsible for individually executing load balancing policies, thus enabling application-dependent load-balancing policies to be implemented, and *server-side load balancing* [7, 110], where a dedicated load balancer sits between client and server microservices to handle load balancing.

(2) *Approach:* Integrating the effect of the load balancing mechanism and introducing novel load balancing policies to improve the performance of the services is vital in microservice application deployment. References [26, 80] model the end-to-end service latency based on the existing load-balancing policies such as Round Robin and Weighted Round Robin to determine the number of microservice instances to deploy. Meanwhile, some works introduce custom load-balancing policies: Reference [7] proposes a load-balancing policy for a multi-region Fog architecture where the requests are directed based on the residual CPU of each Fog region, Reference [67] uses the variance of the resource occupancy rate of the edge nodes to determine where to direct the requests, Reference [52] uses a meta-heuristic algorithm to place microservice replicas to identify flow paths by considering parameters such as service cost and service latency.

5.3 Networking

Distributed deployment of containerised microservices makes networking one of the integral functions of microservice compositions. This is further complicated by the federation of Fog and Cloud, which results in communication between multiple networking environments and technologies. Reference [33] uses Kubernetes to handle the networking among microservices instances, whereas Reference [67] integrates Istio[6] service mesh framework to handle inter-service communication. Reference [110] integrates Kubernetes with KubeEdge for the edge network. To overcome the limitation Kubernetes network model and assign subnets per host, Reference [7] uses Flannel, a **Container Network Interface (CNI)** and Istio on top of Kubernetes orchestration. Reference [15] explores and compares multiple CNIs, including Flannel, Weave, Calico, and OVN. Reference [24] uses MQTT Broker, an asynchronous messaging-based communication mechanism to transmit messages over the network among decoupled microservices.

5.4 Elasticity

Elasticity indicates the ability of the microservices to be dynamically scaled up and down dynamically in a performance-aware manner. With the use of lightweight deployment technologies such as containers, microservices can be easily auto-scaled to improve the performance of the application while ensuring optimum resource utilisation. Out of the many works that consider horizontal scalability of microservices, the majority consider this during the initial placement of the application to make use of resource-constrained Fog devices but fail to use auto-scaling/elasticity under dynamic changes in the environment (i.e., load changes, failures, etc.). Dynamic placement algorithms proposed in References [67, 69, 79] consider elasticity. However, we can further analyse it based on the supported type of elasticity: scaling up, scaling down, and the method of scaling: horizontal, vertical. Reference [79] considers both vertical and horizontal scaling but only considers scaling up as new user requests arrive. Reference [67] proposes a utilisation threshold-based policy to horizontally scale up/down microservices through continuous monitoring of the resources. Practical frameworks and simulators [33, 69] provide the infrastructure required to auto-scale (both up and down, horizontal and vertical) microservices through customised policy implementations.

5.5 Monitoring

Monitoring is the collection of application and platform metrics in such a way that they can be used to detect failures, performance degraded states, and so on, and act accordingly to maintain system performance. The highly dynamic nature of containerised microservices makes monitoring a challenging task, which has resulted in the development of open-source monitoring tools that can

---

[6]https://istio.io/.

handle the large volume of moving parts in microservices-based application deployment. Hence, References [7, 33, 67] integrate Prometheus[7] to their orchestration platform to monitor multiple platform metrics (i.e., request number, response times, resource consumption, network communications, etc.). Meanwhile, References [24, 110] implement their own customised monitoring tools to monitor metrics across Edge-Cloud integration.

5.6 Other

Other composition-related tasks include fault tolerance and security. Reference [61] proposes a conceptual framework where API Gateway handles the responsibility of fault-tolerance by data recovery, service re-composition, and re-submission of a failed request. Reference [24] introduces three main components: health check component, circuit breaker, and timeout component to identify and isolate failures to avoid cascading failures under MSA. Reference [24] implements centralised security components to manage authorisation and authentication required for microservice access.

5.7 Research Gaps

Based on the analysis of existing works presented in Table 4, we identified the following gaps:

(1) Current works demonstrate less emphasis on load-balancing policies and their effect on the formulation of the placement problem.
(2) There is scope for improvement in elasticity (both horizontal and vertical scaling), fault tolerance, and microservice security through proper monitoring of the deployed application at the application and platform level.
(3) Effect of orchestration components (i.e., service registry, API gateways, load balancers) on the application performance needs to be analysed and demonstrated in terms of handling their possible failures, delay overheads, and so on.

6 PERFORMANCE EVALUATION

Accurate policy evaluation is one of the vital steps in designing novel placement algorithms for fast-evolving fields such as IoT and Fog computing. To this end, we propose the final taxonomy by analysing the crucial aspects of the evaluation phase, as shown in Figure 9. Afterwards, current works are mapped to the taxonomy to identify gaps and possible improvements (see Table 5).

6.1 Evaluation Approach

Evaluation approaches can be categorised as Numerical Evaluations, Simulations, use of Practical test-beds, and Hybrid approaches consisting of combinations of the above methods.

Numerical Evaluations: In numerical experiments, the algorithm is evaluated by numerically calculating specific metrics that provide insights on the fitness of the resultant placement proposed by the algorithm [29, 30, 48, 50] or/and evaluating performance metrics of the algorithm such as execution time, computation complexity, and convergence [43, 48]. References [29, 30] use the Gurobi mathematical optimisation solver to obtain the optimum solution to the formulated MINLP and compare the solution obtained from their proposed heuristic placement algorithm by calculating metrics such as the number of placed applications, network link usage, and so on, numerically. Reference [43] evaluates their approach based on the performance metrics (i.e., the fitness of the best solution, Pareto solution spread, execution time, etc.) of their proposed multi-objective evolutionary algorithm, whereas Reference [48] analyses the execution time of the algorithm under

---

[7]https://prometheus.io/.
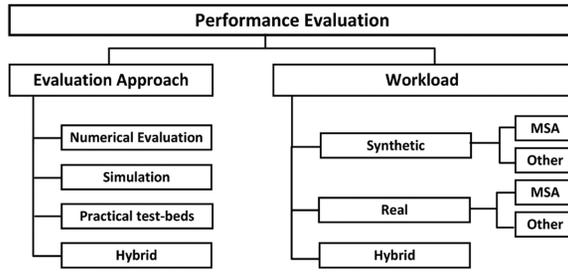
Fig. 9. Taxonomy for performance evaluation of the placement policy.

Table 5. Analysis of Existing Literature Based on the Taxonomy for Performance Evaluation Presented in Figure 9

Work	Evaluation Approach			Workload			Work	Evaluation Approach			Workload		
	Numerical	Simulation	Practical	Synthetic	Real	Hybrid		Numerical	Simulation	Practical	Synthetic	Real	Hybrid
[62]		✓(YAFS)		✓(Other)	-	-	[29]	✓	-	✓(FogAtlas)	✓(MSA)	-	-
[79]	-	✓(iFogSim)	-	✓(MSA)	-	-	[43]	✓	-	-	-	-	✓
[30]	✓	-	-	✓(Other)	-	-	[114]	-	✓(C-MATLAB)	-	✓(MSA)	-	-
[26]	-	✓ (MATLAB + CloudSim)	-	✓(MSA)	-	-	[103]	-	✓(C-MATLAB)	-	✓(MSA)	-	-
[80]	-	✓(iFogSim)	-	✓(MSA)	-	-	[7]	-	✓(C-Python)	✓	-	-	✓
[28]	-	✓(iFogSim)	-	✓(MSA)	-	-	[91]	-	✓(ND)	-	✓(MSA)	-	-
[67]	-		✓	-	✓(MSA)	-	[1]	-	-	✓	-	-	✓
[109]	-	✓(C-Java)	-	-	✓(Other)	-	[52]	-	✓(ND)	-	✓(MSA)	-	-
[50]	✓	-	-	✓(MSA)	-	-	[48]	✓	-	-	✓(Other)	-	-
[61]	-	✓(C-Python)	-	-	✓(MSA)	-	[4]	-	✓(NS3)	-	✓(MSA)	-	-
[31]	-	✓(C-Python)	-	✓(Other)	-	-	[32]	-	✓(Cloudsim)	-	✓(MSA)	-	-
[45]	✓	-	-	✓(MSA)	-	-	[88]	-	✓(iFogSim)	-	-	✓(MSA)	-
[36]	-		✓	-	✓(MSA)	-	[110]	-	-	✓	-	✓(MSA)	-
[112]	-	-	✓(Astraea)	-	✓(MSA)	-	[44]	-	✓(iFogSim)	-	-	-	✓
[81]	-	✓(iFogSim)	-	-	-	✓	[108]	-	✓(C-C++)	-	✓(MSA)	-	-
[40]	-	✓(iFogSim)	-	✓(Other)	-	-	[96]	✓	-	-	✓(MSA)	-	-
[64]	-	✓(ND)	-	✓(MSA)	-	-	[11]	-	✓(C-Python)	-	✓(MSA)	-	-

different experimental settings to evaluate the scalability of the algorithm, thus deriving the system size the algorithm can handle.

Simulations: For the evaluation of microservices-based application placement in Edge/Fog computing environments, both open-source simulators (i.e., iFogSim [28, 44, 79, 80], YAFS [62], CloudSim [26, 32]) and custom simulators [7, 103, 114] are used by the current works. Simulators such as iFogSim [69] and YAFS [63] provide in-built microservice-related features such as distributed application modelling, service discovery, and load balancing, thus enabling the users to simply implement their placement policy within the simulator or implement the algorithm separately (e.g., MATLAB [26], IBM CPLEX [80]) and input the resultant placement to the simulator for evaluations.

Practical: To evaluate placement algorithms using practical frameworks, Reference [29] uses "FogAtlas," an open source framework for microservices deployment and orchestration in Fog environments, Reference [112] implements a framework called "Astraea" for the management of GPU microservices, whereas References [1, 7, 67, 110] implement customised test-beds with functionalities relevant to the placement algorithms. They use popular container-orchestration platforms such as Docker swarm [1], Kubernetes [7, 67], KubeEdge [110] in their implementations.

Hybrid: Some of the works use multiple evaluation approaches to analyse and evaluate the proposed placement policies from multiple perspectives. Reference [7] uses both Simulations and practical test beds for evaluation. Simulations carry out large-scale experiments, whereas test beds further verify the results of the simulations by carrying out a selected set of experiments. Reference [80] uses a combination of numerical evaluations and simulations where numerical evaluations are used to improve and fine-tune the meta-heuristic placement algorithm, whereas the simulation evaluates the resultant placements.

6.2 Workload

For the analysis of the workload, we consider the nature of the application placement requests used to evaluate the placement policy. We can categorise them as Synthetic, Real, or a combination of both, denoted as Hybrid.

Synthetic: Synthetic workloads are created either by mimicking specific microservices-based applications [29, 32, 79] (categorised in the taxonomy as MSA) or using generic application models such as DGs or DAGs [30, 31, 48, 62] (categorised in the taxonomy as Other) to generate a workload consisting of multiple applications with heterogeneous resource and QoS requirements. Reference [29] models the applications following a microservices-based IoT application for face recognition consisting of two chained microservices. Reference [79] models a smart-healthcare application and creates a synthetic workload based on the modelled application. Reference [32] models smart city and forest surveillance applications. Meanwhile, References [30, 31, 48] generate random synthetic DAGs as microservices-based applications, whereas Reference [62] uses **Growing Network (GN)** graph structure where graphs are created by adding nodes one at a time to existing nodes to develop microservices-based applications following Directed Graphs as the interaction pattern.

Real: Real workloads include the use of already implemented microservices-based applications (categorised in the taxonomy as MSA) or adapting performance traces of applications that follow other application models (categorised in the taxonomy as Other). Reference [67] uses Bookinfo,[8] an online book store application following MSA along with the hotel reservation booking application from DeathStartBench[9] [38], which is a benchmark application suite following MSA. Reference [36] also uses benchmark applications from DeathStartBench along with the benchmark application provided in Reference [115]. Reference [112] uses AI-based GPU microservices available in AIbench[10] to create the workload. Reference [61] uses the curated dataset available in Reference [82], which consists of 20 open-source projects based on MSA. In contrast to the above examples, Reference [109] uses traces from Google Cluster 2019 [98]. These traces provide data on task requests (i.e., CPU. memory, deadline, etc.), and as Reference [109] models microservices as independent components that do not interact with other microservices, the said dataset is easily adapted by this work for evaluations.

Hybrid: References [43, 44] use a microservices-based e-commerce application known as Sock Shop[11] provided under Apache License 2.0, along with two other synthetic application models (an online EEG tractor beam game and intelligent surveillance application) to create the workload. Reference [7] creates a synthetic workload for simulation-based studies and implements a microservices-based application name "Paper Miner" designed for mining research papers and deploys it on a real-world platform to evaluate the placement algorithm.

6.3 Research Gaps

Based on the analysis of existing works presented in Table 5, we identified the following gaps related to the evaluation of placement algorithms developed for microservices-based application placement within Fog environments:

(1) Lack of use in practical test beds is one of the prominent drawbacks of currently used evaluation approaches. The majority of the available works use numerical evaluations or simulations to evaluate the performance of their placement policies but fail to validate them on

---

[8]https://istio.io/latest/docs/examples/bookinfo/.
[9]https://github.com/delimitrou/DeathStarBench/tree/master/hotelReservation.
[10]https://www.benchcouncil.org/aibench/index.html.
[11]https://microservices-demo.github.io/.

real test beds. Thus, overheads related to orchestration tasks (i.e., service discovery, load balancing, auto-scaling, etc.), failure characteristics, resource contention among microservices, and so on, are not accurately captured. Moreover, the suitability of the Edge/Fog devices to act as the placement engine that runs the algorithms is not evaluated in practical settings.

(2) As Edge/Fog computing paradigms are still relatively new and yet to be adopted by the service providers, simulators play a significant role in evaluating placement policies. However, this requires a standard open-source simulator for use among the research community and continuous improvements through collaboration. As microservices-based application placement in Fog environments is still in its infancy, we see the increased use of custom simulators due to the lack of open-source simulators that capture all related aspects of microservice orchestration.

(3) Lack of real-world traces or actual implementations of applications for the deployment within test beds is another significant gap in current research. The existing benchmark applications do not include IoT applications, making it harder to capture their characteristics accurately.

7 FUTURE RESEARCH DIRECTIONS

Based on the research gaps identified in previous sections, we propose future directions for microservices-based IoT application placement in Fog environments.

Dynamic placement algorithms - To maintain application QoS under the dynamic nature of the Fog infrastructure and fluctuating workloads, application management algorithms must adapt and make decisions accordingly. Online placement algorithms that carry out continuous re-evaluation of the application placements can achieve this. Algorithms can exploit the independently deployable nature of the microservices, which adds dynamic behaviour to them through auto-scaling, migration, proactive redundant placements, Fog-Cloud balanced usage, and so on. To this end, the placement techniques can benefit from AI-based techniques, such as evolutionary algorithms and ML techniques, such as Reinforcement Learning, that can adapt to dynamic environmental changes.

Placement within federated multi-fog multi-cloud environments - Federation among Fog resources provided by multiple Fog infrastructure providers (multi-fog) and multi-cloud environments is emerging as an approach better suited for utilising geo-distributed and resource-constrained Fog computing resources to meet the non-functional requirements of IoT applications. The loosely coupled nature of the microservices allows them to span across such environments while maintaining seamless connectivity among them. However, for such scenarios, placement policies need to consider costs, resource availability, security, composition-related overheads, and limitations based on the infrastructure provider and location of the Fog resources. Moreover, placement policies must evolve towards distributed placement approaches to handle placement within multi-fog environments.

Microservice orchestration platforms for Fog - For the evaluation of scheduling policies within Cloud environments, commercial platforms such as AWS, Google Cloud, and Microsoft Azure are available. As Fog computing is still in its early stages of industrial adaptation, current research uses small, custom-built test beds. However, they lack support for large-scale experiments, thus failing to capture important aspects related to MSA, such as distributed, location-aware deployments, load balancing, reliability, security, and interoperability of services within the large-scale IoT ecosystem. Moreover, they should reflect novel technologies (i.e., container orchestration, service mesh, monitoring tools, overlay networks, etc.). Hence, scalable and extensible container orchestration platforms for Fog environments should be implemented for research purposes.

IoT workloads/benchmarks related to MSA - Lack of microservices-based IoT workload traces from large-scale deployments and benchmark IoT applications that follow MSA hinder the evaluation of placement policies considerably. Enterprise workload traces of IoT applications can be used to derive accurate data related to request volumes, patterns, diversity in usage of services, and so on. Collecting such data over a long time within large-scale IoT deployments and making them accessible to the research community is significant for accurately evaluating placement algorithms.

Security-aware placement - Security-aware placement of microservices must consider both security guarantee levels of microservices and secure data transmission among microservices [106]. Security levels of each microservice play an important role in the overall security of the application services accessed by the users. Due to the availability of multiple candidate container images of microservices providing the same functional requirements but with different security implementations (i.e., authenticity, trustworthiness, security techniques, etc.), placement algorithms can be improved to consider microservice security during placement to ensure the expected level of dependability of application services. Moreover, Data privacy is one of the main concerns of data-driven IoT applications. Distributed deployment of microservices across Fog-Cloud, along with the vulnerability of open microservice interfaces, poses a considerable security threat to sensitive data transmission and processing. The independently deployable nature of microservices enables the migration of microservices easily across federated Fog environments and between Fog and the Cloud. However, placement algorithms must incorporate data privacy and security threats related to such migrations in making deployment decisions.

Resource contention handling - Due to the concurrent execution of multiple containers within the same device, resource limitation in the devices and complex interaction patterns of the microservices, resource contention among microservices can affect application performance negatively. The development of intelligent algorithms that can proactively identify resource contentions during microservice placement, dynamically auto-tune container parameters, or migrate containers across the Fog-Cloud continuum has the potential to mitigate this challenge.

Observability and monitoring driven maintenance - Deployment of microservices-based IoT applications creates a distributed system of many microservice instances that can be dynamically created and destroyed. Observability and monitoring can be used to detect performance anomalies within such systems. This requires distributed tracing, monitoring, and analysis of the system at both application and platform levels, which would create massive amounts of data of different metrics, logs, and traces. This poses a big data analysis challenge where data mining and artificial intelligence models can be integrated with the placement policy to make performance-aware decisions in handling performance anomalies and failures within the system.

Placement within NFV-enabled networks - **Network Function Virtualisation (NFV)** has become a key enabler for 5G and 6G mobile networks. NFV virtualises network functions (i.e., firewalls, routers, load balancers, etc.) to provide flexible management and orchestration of network resources. Thus, NFV creates research opportunities to develop end-to-end zero-touch management mechanisms for network and IoT application services to ensure QoS requirements of large and fluctuating IoT traffic volumes. Moreover, virtualised network functions are developed as containerised microservices, which has resulted in virtualised network functions such as firewalls and load balancers to become a part of the microservice-based composite application services (known as service function chains). This requires microservices-based application placement and request routing to be solved in the context of service function chains, where the dynamic placement of containerised network functions becomes a significant part of microservices-based IoT application

placement to improve dynamic traffic routing, security, and so on, to satisfy the non-functional requirements of the application services.

Fault-tolerant placement and management of microservices - MSA avoids a single point of failure by decomposing applications into loosely coupled microservices. While this improves fault tolerance, it creates more points of failure and also cascading failures due to interacting microservices. Root-cause tracing, predictive redundant placements, awareness of microservice-level stability patterns (i.e., circuit breaking, timeouts, retries, etc.) and their effect can be used to generate more resilient placement, migration, and request load-balancing approaches to improve fault-tolerance of IoT application services.

Availability assurance under continuous integration and delivery - One of the main reasons for the rising popularity of MSA is its ability to support rapid development and deployment cycles to keep up with the fast-evolving IoT domain. However, this requires smooth integration and deployment of novel updates and changes while minimising service downtime. MSA handles this through multiple deployment strategies such as Canary deployments, Rolling deployments, Blue-Green deployment, and A/B testing [17, 37, 49]. To support continuous integration and delivery requirements in an availability-aware manner, continuous placement policies need to be developed, including combinations of multiple deployment strategies supported by MSA.

8 SUMMARY

In this article, we discussed IoT applications designed and developed using MSA and their placement within Fog computing environments. We conducted a comprehensive background study and identified four critical aspects of microservices-based application placement: modelling of MSA, placement policy creation, microservice composition, and performance evaluation. We proposed taxonomies for each of the aspects, highlighting features related to MSA. Moreover, we analysed and discussed the current literature under each taxonomy and identified research gaps. Finally, we compiled future research directions to improve the placement of microservices-based IoT applications in Fog computing environments.

APPENDIX

A BACKGROUND

This section presents a comprehensive background on MSA, IoT applications, and different computing paradigms such as Edge computing, Fog computing, Mobile Edge Computing, and Osmotic Computing to highlight their characteristics. Moreover, this section compares microservices-based application placement and the traditional DAG-based workflow scheduling to highlight the differences between the two problems.

A.1 Comparison between Microservice and Monolithic Architecture for Applications

An enterprise application usually consists of a server-side application that implements its domain-specific business logic, client-side user interfaces supporting different clients such as desktop browsers and/or mobile browsers, and a database to persist the data. Moreover, the application may connect with other third-party applications (through web services or message brokers) and expose APIs for third parties to consume. The design and development of such applications follow different architecture patterns, depending on the complexity of the business domain. Figure 10 presents a general representation of an application developed using Monolithic and Microservice architecture.

(a) Monolithic application architecture   (b) Microservice application architecture (MSA)
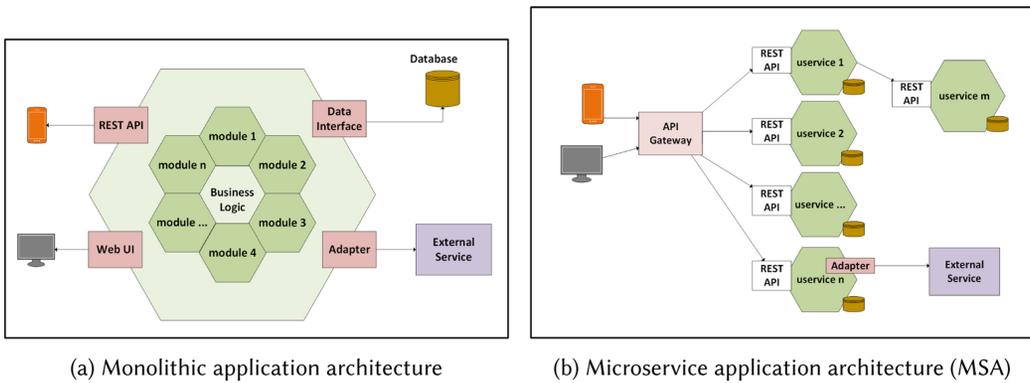
Fig. 10. Application architecture (Monolithic vs. Microservices).

In monolithic architecture, the server-side application is a single logical executable that handles HTTP requests from the client side, performs logical operations, communicates with the database, and populates HTML views to be displayed on the client side. The server-side application can be designed and developed either as a single process or a modular monolith where modules invoke each other through method/function calls at the programming language level [76]. Using monolithic architecture is advantageous during the early stages of an application or if the application is quite simple with only a few core functionalities. It provides a better approach to defining requirements and building, testing, and deploying a working product quickly and efficiently.

But as the applications grow, they tend to outgrow the monolithic architecture [86]. As all modules of the application are combined into a single unit, the coupling among them increases, and the application becomes too complex to understand, which makes it hard to make changes and improvements to the application. As any modification to the application affects the entire application, end-to-end testing becomes complex, resulting in high deployment delays. Moreover, monolithic architecture lacks flexibility, as it requires a single development stack that hinders the adaptation of new technologies or heterogeneous technologies depending on the functionality of each module. Monoliths also lack reliability, because a fault in one place results in the failure of the entire application. Moreover, different modules inside a monolith may have different resource requirements, which makes the application challenging to scale. Thus, monolithic architecture lacks support for agile development and limits the rapid growth of applications.

MSA aids in overcoming the limitations of the monolithic architecture within rapidly growing software ecosystems. Martin Fowler defines MSA as "an approach to developing a single application as a suite of small services, each running as independent processes and communicating with lightweight mechanisms, often an HTTP resource API" [34]. Figure 10(b) depicts a microservice application developed using the API design pattern where all services are exposed to clients through an API gateway that directs client requests toward microservices. MSA is an evolution of **Service Oriented Architecture (SOA)**, where microservices are fine-grained compared to services in SOA. It uses dumb pipes for the communication among them, unlike in the case of SOA, where services communicate using smart pipes containing business and message processing logic [34].

Thus, microservices are independently deployable and scalable units that are designed around business logic adhering to the single responsibility principle and contain their own database for persisting data [57]. Moreover, microservices are loosely connected components that can be easily integrated to create complex applications. From the software design and development perspective, this allows small teams to work on each service with minimum dependencies among the

Table 6. Comparison between Monolithic and Microservice Architecture

Monolithic Architecture	Microservice Architecture
Server-side application is a single logical executable	Server-side consists of independently deployable, multiple microservices
Modules communicate through language-level method invocations	Inter-process communication through REST APIs or lightweight messaging
Low flexibility	Flexible - different programming languages and technologies can be used based on microservice requirements
Less support for scaling	Highly scalable (independently deployable and scalable microservices)
Longer time from development to deployment	Supports rapid and agile development and deployment
Unreliable due to single point of failure	Higher resilience to failure due to failure isolation and redundancy support

teams, thus increasing development velocity by supporting independent development and testing of microservices. Due to fine-grained modularity, each microservice can be deployed on hardware that best matches its resource requirements (i.e., CPU-intensive, memory-intensive, I/O-intensive, etc.) and can be deployed and scaled independently according to the load of each service. Unlike monolithic architecture, MSA has better fault isolation, as a fault in a certain service only affects that service. Furthermore, the independent scalability of microservices improves redundant deployment to achieve fault tolerance. As services are loosely coupled, new technologies that best suit the microservice can be easily adapted. This mitigates the need to stick to a single technology stack and provides the flexibility to evolve with technologies. Table 6 summarises and compares the characteristics of the two architectures.

MSA enables continuous development and deployment of rapidly evolving complex applications but introduces novel challenges in design, development, deployment, and orchestration, described in detail below:

- **Microservice design:** From a software design perspective, defining microservice boundaries can be identified as one of the main challenges of MSA. The level of granularity directly affects the performance of the applications. Dividing the application logic into too many microservices increases the number of remote calls made among microservices, thus increasing the latency of the operations. In contrast, less granularity creates macro modules with lower cohesion and higher coupling. Hence, an optimum level of granularity needs to be achieved by using the domain-driven approach for the separation of business concerns so extensibility and adaptability are supported as the application requirements evolve [47, 93]. Moreover, it results in designing microservices as independently deployable and scalable components.

- **Application Placement:** Granularity creates complicated dependencies among microservices and complex composition patterns, which results in composite services. Openness in microservice design allows reusability of microservices between multiple composite services with heterogeneous QoS requirements [77], conditional branching, which results in alternative dataflows within composite services [78], and use of third-party microservices [3]. These characteristics need to be considered during the application placement to improve performance (i.e., QoS, resilience, scalability, security, etc.) while satisfying competing performance requirements of the services. Moreover, characteristics of the microservices, such as their independently deployable and scalable nature, have the potential to improve performance if appropriately utilised by the placement algorithms. Thus, novel placement algorithms are required to handle the above complexities efficiently while using the benefits provided by the architecture [79, 101].

- **Application management:** The increase in the number of moving components per application makes management more complex. At the same time, due to the elasticity of microservices, service instances can be dynamically scaled up and down with the changing workload or system failures. This requires support for automated deployment using orchestration platforms. To overcome the complexities introduced by the number of microservices

and their dependencies, proper monitoring is crucial to provide insights (i.e., the health of the microservices/APIs, isolation of failures, demand variations, etc.) to make management decisions.
- **Microservices communication:** With the development of applications as a collection of interconnected modules, proper communication among microservices must be established using different communication patterns. Suitable communication patterns are selected by analysing the nature of the interactions among microservices [93]. Such patterns include Synchronous communication (i.e., request-response methods like HTTP-REST), Publish-subscribe and Asynchronous communication using message queues (RabbitMQ, Apache Kafka, and Apache ActiveMQ [23]).
- **Service discovery:** Within a microservices application, microservices communicate with each other to perform tasks. This requires client microservices to be aware of other available microservices that can be invoked to perform specific tasks. With the distributed nature of the microservices and the dynamism of microservices placement (auto-scaling, migration, service failures), service discovery has become one of the main challenges in using MSA. Dynamic service discovery involves checking the status of existing microservices and detecting and registering new microservices as they become available in real-time, with minimum overhead.
- **Load balancing:** MSA supports horizontal scalability or a combination of horizontal and vertical scalability of the microservices. As a result, to ensure performance, load balancing is required to direct requests among horizontally scaled microservices. Efficient load balancing algorithms have to be designed to minimise the networking and computation overhead and ensure microservice availability through load analysis, link analysis, and forecasting to adapt to dynamic scenarios [77, 102]. Thus, load balancing is tightly coupled to service discovery and monitoring of the microservices-based systems.
- **Fault tolerance:** Even though MSA supports higher fault isolation compared to monolith architecture, it still requires fault tolerance methods to handle failures. Having a large number of components means there are more points of failure. Moreover, communication failures among components become a potential cause as well. Fault-tolerance mechanisms should be able to tackle such failures and avoid cascading failures as well. To achieve this, MSA aims to handle failures at both the development and deployment phases. Microservice applications integrate methods such as circuit breakers, retries, timeouts, and deadlines to the microservice design to minimise the effect of failures, whereas redundant placement, checkpoint and restore, migration, and so on, are used at deployment to improve the resilience and availability of the application under failures.

A.2 Internet of Things (IoT) Applications and their Characteristics

Internet of Things (IoT) refers to connected objects that collect and share data with other devices over the internet [56, 65]. It transforms conventional physical objects into smart objects. With the software/hardware improvements of the connected devices along with bandwidth and latency improvements in telecommunication technologies such as 5G/6G and other wireless technologies like Wifi6, the number and variety of connected devices are increasing rapidly [18, 19, 46]. Cisco predicts that 500 billion devices will be connected to the internet by 2030, which would result in the generation of a massive amount of data that can be used to generate insights in a variety of IoT application domains such as smart healthcare, smart city, smart home, **Industrial IoT (IIoT)**, smart agriculture, and smart transportation [54, 116].

Due to rapid growth in IoT data generation, applications have to undergo fast design, development, and deployment cycles to both improve existing IoT services (i.e., moving to new tech stacks,

AI augmentation and updating ML models, integration of novel data types, improvements in data processing method/algorithms, etc.) and introduce new services to the end-users. The ultimate vision of IoT is to move away from application development as "point solutions" and evolve towards an "IoT ecosystem" where cross-platform integration of heterogeneous devices, data, communication technologies, and application services occur [25, 107]. To this end, IoT applications should be flexible to change, support service re-usability, and expose open interfaces to enable third-party integration of application services (i.e., big data analytic services, payment gateways, authentication and authorisation services, data visualisation services, etc.) [84].

Due to improvements in **Radio Access Network (RAN)** technologies and consequent growth in mobile IoT devices (i.e., wearable gadgets, smartphones, vehicles, etc.), mobile IoT workloads keep increasing rapidly. To accommodate such fluctuating workloads, IoT application design has to mainly focus on the scalability of the application deployment [100]. Moreover, IoT applications consist of a large number of diverse services in terms of suitability of development technology stacks (i.e., I/O intensive services using NodeJs, machine learning using Python, etc.) and QoS expectations (i.e., bandwidth-hungry services like **Multimedia Internet of Things (M-IoT)**, latency-sensitive applications such as healthcare, tactile internet, etc.).

Due to these characteristics and requirements of IoT applications, their development is moving away from Monolithic architecture towards distributed application architectures, whereas deployment is moving away from cloud-centric deployment towards distributed computing paradigms like Fog computing.

A.3 Edge and Fog Computing, Osmotic Computing and Related Paradigms

Fog computing is a novel network computing paradigm that extends cloud-like services towards the edge of the network [68]. The fog computing paradigm was first introduced by Cisco in 2012 as a platform to support the unique requirements of IoT, such as low latency, location awareness, mobility support, and geo-distribution [10]. To this end, Fog computing introduces an intermediate layer between IoT devices and Cloud data centres [68], which is organised in a multi-tier architecture. It exploits computation, storage, and networking resources that reside within the path connecting end devices to the Cloud data centres [85]. Thus, Fog resources consist of a diverse set of resources (e.g., smart routers and switches, personal computers, edge servers, Raspberry Pi devices, micro-datacentres, cloudlets, etc.) that are distributed and resource-constrained compared to Cloud data centre resources. To overcome the resource limitations, the Fog computing paradigm maintains federated Fog computing architectures, where distributed Fog resources collaborate to satisfy client requirements [5]. Moreover, the Fog computing layer maintains a seamless connection with the Cloud [111] so computation-intensive tasks can be carried out using Cloud resources.

Edge computing and **Mobile Edge Computing (MEC)** also follow a similar concept to Fog computing, where their main objective is to move computation toward the users [70]. References [68, 70, 111] define Edge computing as a paradigm limited to the edge network, which consists of devices such as mobile phones, access points, and so on, at the immediate first hop from IoT devices. In contrast, Fog extends this concept further to include the core network along with the integration of Cloud data centres when necessary to provide IaaS, PaaS, and SaaS services in the proximity of the data sources. MEC focuses on mobile end-users and moves processing and storage capabilities to the edge of the mobile network by improving the capabilities of the base stations that reside within the **Radio Access Network (RAN)** of the 5G/6G networks [27]. However, all these paradigms share common characteristics, such as heterogeneity of resources and limited and distributed resource availability, along with main goals such as lower latency, lower network congestion, and mobility support. Figure 11 presents a comparison among Fog, Edge and MEC architectures.
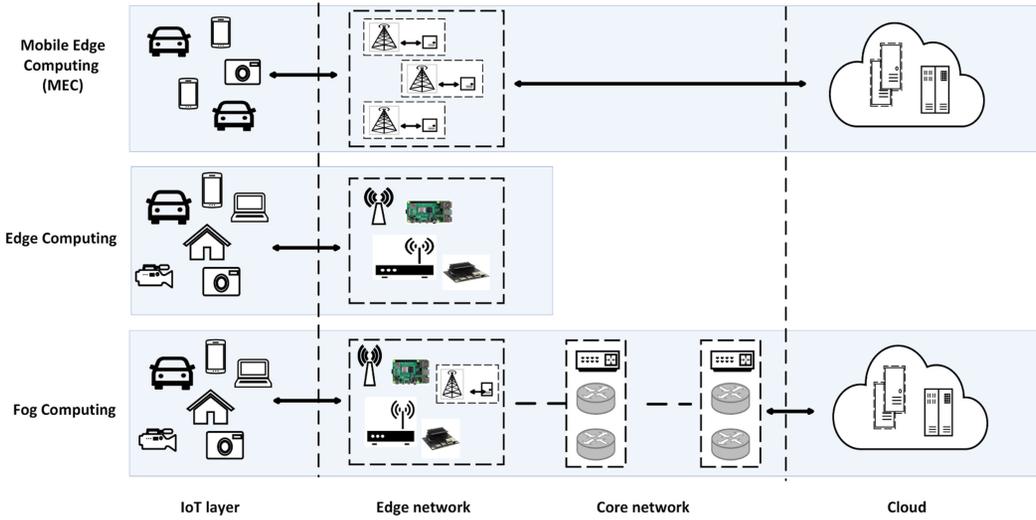
Fig. 11. Fog computing, edge computing, and mobile edge computing (MEC).

Osmotic computing is a relatively new paradigm, first introduced in Reference [101], especially focusing on microservices-based application placement within Fog and Cloud environments. Osmotic computing aims to achieve a balanced deployment of microservices by utilising Edge/Fog and Cloud resources to satisfy the requirements defined at different levels of the microservices-based IoT application (i.e., service-level QoS requirements, microservice-level resource requirements, etc.). Osmotic computing identifies MSA as a suitable approach for developing IoT applications for deployment within integrated Edge/Fog and Cloud environments due to their fine granularity, fast deployment, and elasticity. Thus, this paradigm tries to address microservices-specific challenges such as microservice orchestration, networking, monitoring, elasticity control, and dynamic placement [6, 101].

A.4 Microservices-based IoT Applications Placement vs. DAG Workflow Scheduling

In this section, we compare the "Microservices-based IoT application placement" problem with the **Directed Acyclic Graph (DAG)** workflow scheduling problem [87], which is extensively addressed in the existing literature, considering both Fog and Cloud environments. We identify similarities and differences between the two to highlight the unique aspects related to the Microservices-based IoT application placement problem. Microservices-based IoT application placement problem and summarise them in Table 7.

DAG-based workflows are used to model business processes as collections of distributed tasks where tasks, their dependencies, and dataflows are represented using DAGs [87]. Each DAG is characterised by a tuple, $\{V, E\}$, where V and E denote vertices and edges, respectively. In DAG-based workflows, vertices represent the tasks or sub-processes of the main process, and directional edges depict the dataflow from the source task to the destination task, where the output from the prior task is fed as input for the next task. Thus, each DAG would consist of one or more starting tasks with no incoming edges and an ending task that provides the processed output.

As microservice applications consist of interconnected microservices collaborating to perform domain logic, existing works model such applications as DAGs [31, 43], where microservices are represented by the vertices (V) of the DAG, and edges (E) represent the dependencies among microservices with direction from client microservices to invoked service. Thus, the directional edges

Table 7. Comparison between Microservices-based Application Placement vs. Directed Acyclic Graph (DAG) Workflow Scheduling

| Microservices-based IoT applications Placement	DAG workflow Scheduling
Microservice invocation can be represented by a DAG	Invocation and dataflow are represented by a DAG
Multiple end-user services per app	Each DAG represents a single service
Microservices shared among services/ applications	No shared tasks among workflows
Per-service QoS with multiple services per app	QoS per workflow/ensemble of workflows
Life cycle: perpetual	Life cycle: mostly ephemeral
Service access shared among a large number of users	Mostly focus on a particular user
Complex interactions (shared microservices, candidate microservices, third-party microservices)	Not a main focus
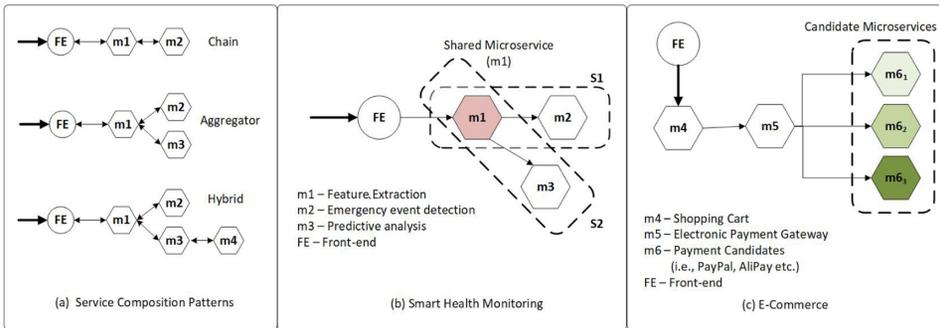
Fig. 12. Characteristics of microservices-based composite services.

denote microservice invocations. However, unlike in the case of DAG workflows, dataflows in microservice applications can not always follow a DAG due to the complex data dependencies among microservices, which can create cyclic dataflows. This can be further explained through the concept of composite end-user services in MSA. In IoT applications, "services" can be defined as functionalities accessed by the end-user expecting a certain output, and each application can consist of one or more such services [80]. Fine-granularity of MSA creates composite services with different dataflow patterns such as chained, aggregator, and hybrid [83] (Figure 12(a)). Such patterns can result in cyclic dataflows where one microservice invokes other microservices, receives responses from them, and finally aggregates and/or further processes the responses before returning the results of the composite service.

Moreover, each IoT application consists of multiple composite services with heterogeneous QoS requirements. As an example, in a smart healthcare application for patient monitoring, there can be multiple services, such as emergency notification services that detect abnormalities in measured vitals in real-time (S1 in Figure 12(b)) and latency-tolerant services for predictive analysis to give early warnings about possible health issues to prompt preventive actions (S2 in Figure 12(b)) [80]. Herein, large-scale IoT application placement has to consider competing QoS requirements of its services. These services can share some microservices among them, which further complicates the placements decisions (in Figure 12(b) Feature Extraction microservice - m1 is part of both S1 and S2). In contrast to this, workflow scheduling either defines QoS per workflow or an ensemble of workflows.

Each large-scale IoT application has to support a large user base of distributed users that expect ubiquitous access to the application. Thus, application placement includes challenges related to the application's scalability based on the number of users, their access locations, and time. Moreover, the life cycle of the placement is also perpetual. In contrast to this, workflow scheduling is defined from a user perspective, where each user makes the placement request for their particular

use, and workflow deployment exists until the request from that client is processed, which makes the life cycle ephemeral. Moreover, MSA creates complex interaction patterns such as shared microservices among composite services, candidature microservices resulting in conditional dataflows based on the users, and third-party microservice usage. Reference [114] discusses electronic payment microservices where users can select one out of many payment options (e.g., PayPal, AliPay, etc.) and the requests are directed to the microservice relevant to the selected option (Figure 12(c)). This results in multiple possible data paths within a single service and different levels of demand for each payment microservice based on the composition of the users accessing the service. Existing works on workflow scheduling rarely consider such interactions.

Even though microservice applications can be modelled as DAGs, the above-highlighted characteristics differentiate microservices-based application placement problem from existing research focusing on DAG-based workflow scheduling within Fog environments.

REFERENCES

[1] Muhammad Abdullah, Waheed Iqbal, Arif Mahmood, Faisal Bukhari, and Abdelkarim Erradi. 2020. Predictive autoscaling of microservices hosted in fog microdata center. *IEEE Syst. J.* 15, 1 (2020), 1275–1286.

[2] Madhura Adeppady, Carla Fabiana Chiasserini, Holger Karl, and Paolo Giaccone. 2022. iPlace: An interference-aware clustering algorithm for microservice placement. In *Proceedings of the IEEE International Conference on Communications*. IEEE, 5457–5462.

[3] Eyhab Al-Masri. 2018. Enhancing the microservices architecture for the internet of things. In *Proceedings of the IEEE International Conference on Big Data (Big Data'18)*. IEEE, 5119–5125.

[4] Derian Alencar, Cristiano Both, Rodolfo Antunes, Helder Oliveira, Eduardo Cerqueira, and Denis Rosário. 2021. Dynamic microservice allocation for virtual reality distribution with qoe support. *IEEE Trans. Netw. Serv. Manag.* 19, 1 (2021), 729–740.

[5] Abdullah M. Alqahtani, Barzan Yosuf, Sanaa H. Mohamed, Taisir E. H. El-Gorashi, and Jaafar M. H. Elmirghani. 2021. Energy minimized federated fog computing over passive optical networks. In *Proceedings of the International Symposium on Networks, Computers and Communications (ISNCC'21)*. IEEE, 1–6.

[6] Darko Andročec. 2019. Systematic mapping study on osmotic computing. In *Proceedings of the Central European Conference on Information and Intelligent Systems*. Faculty of Organization and Informatics Varazdin, 79–84.

[7] Valentino Armani, Francescomaria Faticanti, Silvio Cretti, Seungwoo Kum, and Domenico Siracusa. 2021. A cost-effective workload allocation strategy for cloud-native edge services. *arXiv preprint arXiv:2110.12788* (2021).

[8] D. Baburao, T. Pavankumar, and C. S. R. Prabhu. 2023. Load balancing in the fog nodes using particle swarm optimization- based enhanced dynamic resource allocation method. *Applied Nanoscience* 13, 2 (2023), 1045–1054.

[9] Sanjeevani Bhardwaj and Alok Kole. 2016. Review and study of internet of things: It's the future. In *Proceedings of the International Conference on Intelligent Control Power and Instrumentation (ICICPI'16)*. IEEE, 47–50.

[10] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. 2012. Fog computing and its role in the Internet of Things. In *Proceedings of the 1st Edition of the MCC Workshop on Mobile Cloud Computing*. 13–16.

[11] Wassim Boudieb, Abdelhamid Malki, and Mimoun Malki. 2022. Microservice instances provisioning for IoT applications in fog computing. In *Proceedings of the 5th Conference on Computing Systems and Applications: Advances in Computing Systems and Applications*. Springer, 107–117.

[12] Antonio Brogi, Stefano Forti, Carlos Guerrero, and Isaac Lera. 2020. How to place your apps in the fog: State of the art and open challenges. *Softw.: Pract. Exper.* 50, 5 (2020), 719–740.

[13] Neda Bugshan, Ibrahim Khalil, Nour Moustafa, and Mohammad Saidur Rahman. 2023. Privacy-Preserving Microservices in Industrial Internet of Things Driven Smart Applications. *IEEE Internet of Things Journal* 10, 4 (2023), 2821–2831.

[14] Björn Butzin, Frank Golatowski, and Dirk Timmermann. 2016. Microservices approach for the internet of things. In *Proceedings of the IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA'16)*. IEEE, 1–6.

[15] Alina Buzachis, Antonino Galletta, Lorenzo Carnevale, Antonio Celesti, Maria Fazio, and Massimo Villari. 2018. Towards osmotic computing: Analyzing overlay network solutions to optimize the deployment of container-based microservices in fog, edge and IoT environments. In *Proceedings of the IEEE 2nd International Conference on Fog and Edge Computing (ICFEC'18)*. IEEE, 1–10.

[16] Carmen Carrión. 2022. Kubernetes Scheduling: Taxonomy, Ongoing Issues and Challenges. *ACM Computing Surveys (CSUR)* 55, 7 (2022).

[17] Ramaswamy Chandramouli, Zack Butcher, et al. 2020. Building secure microservices-based applications using service-mesh architecture. *NIST Special Pub.* 800 (2020), 204A.

[18] Chen Chen, Junchao Li, Venki Balasubramaniam, Yongqiang Wu, Yuru Zhang, and Shaohua Wan. 2020. Contention resolution in Wi-Fi 6-Enabled Internet of Things based on deep learning. *IEEE Internet Things J.* 8, 7 (2020), 5309–5320.

[19] Lalit Chettri and Rabindranath Bera. 2019. A comprehensive survey on Internet of Things (IoT) toward 5G wireless systems. *IEEE Internet Things J.* 7, 1 (2019), 16–32.

[20] Breno Costa, Joao Bachiega Jr, Leonardo Rebouças de Carvalho, and Aleteia P. F. Araujo. 2022. Orchestration in fog computing: A comprehensive survey. *ACM Comput. Surv.* 55, 2 (2022), 1–34.

[21] Hans Jakob Damsgaard, Aleksandr Ometov, and Jari Nurmi. 2023. Approximation Opportunities in Edge Computing Hardware: A Systematic Literature Review. *ACM Computing Surveys (CSUR)* 55, 12 (2023).

[22] Antonio De Iasio, Angelo Futno, Lorenzo Goglia, and Eugenio Zimeo. 2019. A microservices platform for monitoring and analysis of IoT traffic data in smart cities. In *Proceedings of the IEEE International Conference on Big Data (Big Data'19)*. IEEE, 5223–5232.

[23] Davi L. de Oliveira, Artur F. da S. Veloso, José V. V. Sobral, Ricardo A. L. Rabêlo, Joel J. P. C. Rodrigues, and Petar Solic. 2019. Performance evaluation of MQTT brokers in the Internet of Things for smart cities. In *Proceedings of the 4th International Conference on Smart and Sustainable Technologies (SpliTech'19)*. IEEE, 1–6.

[24] Cleber Jorge Lira de Santana, Brenno de Mello Alencar, and Cássio V. Serafim Prazeres. 2019. Reactive microservices for the internet of things: A case study in fog computing. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*. 1243–1251.

[25] Flavia C. Delicato, Paulo F. Pires, Thais Batista, Everton Cavalcante, Bruno Costa, and Thomaz Barros. 2013. Towards an IoT ecosystem. In *Proceedings of the 1st International Workshop on Software Engineering for Systems-of-systems*. 25–28.

[26] Shuiguang Deng, Zhengzhe Xiang, Javid Taheri, Mohammad Ali Khoshkholghi, Jianwei Yin, Albert Y. Zomaya, and Schahram Dustdar. 2020. Optimal application deployment in resource constrained distributed edges. *IEEE Trans. Mob. Comput.* 20, 5 (2020), 1907–1923.

[27] Koustabh Dolui and Soumya Kanti Datta. 2017. Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing. In *Proceedings of the Global Internet of Things Summit (GIoTS)*. IEEE, 1–6.

[28] Juan Fang and Aonan Ma. 2020. IoT application modules placement and dynamic task processing in edge-cloud computing. *IEEE Internet Things J.* 8, 16 (2020), 12771–12781.

[29] Francescomaria Faticanti, Francesco De Pellegrini, Domenico Siracusa, Daniele Santoro, and Silvio Cretti. 2019. Cutting throughput with the edge: App-aware placement in fog computing. In *Proceedings of the 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom'19)*. IEEE, 196–203.

[30] Francescomaria Faticanti, Francesco De Pellegrini, Domenico Siracusa, Daniele Santoro, and Silvio Cretti. 2020. Throughput-aware partitioning and placement of applications in fog computing. *IEEE Trans. Netw. Serv. Manag.* 17, 4 (2020), 2436–2450.

[31] Francescomaria Faticanti, Marco Savi, Francesco De Pellegrini, Petar Kochovski, Vlado Stankovski, and Domenico Siracusa. 2020. Deployment of application microservices in multi-domain federated fog environments. In *Proceedings of the International Conference on Omni-layer Intelligent Systems (COINS'20)*. IEEE, 1–6.

[32] Ion-Dorinel Filip, Florin Pop, Cristina Serbanescu, and Chang Choi. 2018. Microservices scheduling model over heterogeneous cloud-edge environments as support for IoT applications. *IEEE Internet Things J.* 5, 4 (2018), 2672–2681.

[33] FogAtlas. Retrieved from https://fogatlas.fbk.eu/.

[34] Martin Fowler and James Lewis. 2014. Microservices a Definition of This New Architectural Term. Retrieved from https://martinfowler.com/articles/microservices.html.

[35] Kaihua Fu, Wei Zhang, Quan Chen, Deze Zeng, and Minyi Guo. 2021. Adaptive resource efficient microservice deployment in cloud-edge continuum. *IEEE Trans. Parallel Distrib. Syst.* 33, 8 (2021), 1825–1840.

[36] Kaihua Fu, Wei Zhang, Quan Chen, Deze Zeng, Xin Peng, Wenli Zheng, and Minyi Guo. 2021. QoS-aware and resource efficient microservice deployment in cloud-edge continuum. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS'21)*. IEEE, 932–941.

[37] Damien Gallagher and Ruth G. Lennon. 2022. Architecting multi-cloud applications for high availability using DevOps. In *Proceedings of the IEEE International Conference on e-Business Engineering (ICEBE'22)*. IEEE, 112–118.

[38] Yu Gan, Yanqi Zhang, Dailun Cheng, Ankitha Shetty, Priyal Rathi, Nayan Katarki, Ariana Bruno, Justin Hu, Brian Ritchken, Brendon Jackson, et al. 2019. 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*. 3–18.

[39] Martin Garriga. 2017. Towards a taxonomy of microservices architectures. In *Proceedings of the International Conference on Software Engineering and Formal Methods*. Springer, 203–218.

[40] Mostafa Ghobaei-Arani and Ali Shahidinejad. 2022. A cost-efficient IoT service placement approach using whale optimization algorithm in fog computing environment. *Expert Syst. Applic.* 200 (2022), 117012.

[41] Mohammad Goudarzi, Marimuthu Palaniswami, and Rajkumar Buyya. 2022. Scheduling IoT applications in edge and fog computing environments: A taxonomy and future directions. *Comput. Sur.* 55, 7 (2022), 1–41.

[42] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. 2013. Internet of Things (IoT): A vision, architectural elements, and future directions. *Fut. Gen. Comput. Syst.* 29, 7 (2013), 1645–1660.

[43] Carlos Guerrero, Isaac Lera, and Carlos Juiz. 2019. Evaluation and efficiency comparison of evolutionary algorithms for service placement optimization in fog architectures. *Fut. Gen. Comput. Syst.* 97 (2019), 131–144.

[44] Carlos Guerrero, Isaac Lera, and Carlos Juiz. 2019. A lightweight decentralized service placement policy for performance optimization in fog computing. *J. Amb. Intell. Humaniz. Comput.* 10, 6 (2019), 2435–2452.

[45] Feiyan Guo, Bing Tang, and Mingdong Tang. 2022. Joint optimization of delay and cost for microservice composition in mobile edge computing. *World Wide Web* 25, 5 (2022), 2019–2047.

[46] Fengxian Guo, F. Richard Yu, Heli Zhang, Xi Li, Hong Ji, and Victor C. M. Leung. 2021. Enabling massive IoT toward 6G: A comprehensive survey. *IEEE Internet Things J.* 8, 15 (2021), 11891–11915.

[47] Sara Hassan, Rami Bahsoon, and Rick Kazman. 2020. Microservice transition and its granularity problem: A systematic mapping study. *Softw.: Pract. Exper.* 50, 9 (2020), 1651–1681.

[48] Xiang He, Zhiying Tu, Markus Wagner, Xiaofei Xu, and Zhongjie Wang. 2022. Online deployment algorithms for microservice systems with complex dependencies. *IEEE Trans. Cloud Comput.* (2022).

[49] Robert Heinrich, André Van Hoorn, Holger Knoche, Fei Li, Lucy Ellen Lwakatare, Claus Pahl, Stefan Schulte, and Johannes Wettinger. 2017. Performance engineering for microservices: Research challenges and directions. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering.* 223–226.

[50] Juan Luis Herrera, Jaime Galán-Jiménez, Paolo Bellavista, Luca Foschini, Jose Garcia-Alonso, Juan M. Murillo, and Javier Berrocal. 2021. Optimal deployment of fog nodes, microservices and SDN controllers in time-sensitive IoT scenarios. In *Proceedings of the IEEE Global Communications Conference (GLOBECOM'21).* IEEE, 1–6.

[51] Cheol-Ho Hong and Blesson Varghese. 2019. Resource management in fog/edge computing: A survey on architectures, infrastructure, and algorithms. *ACM Comput. Surv.* 52, 5 (2019), 1–37.

[52] Tiansheng Huang, Weiwei Lin, Chennian Xiong, Rui Pan, and Jingxuan Huang. 2020. An ant colony optimization-based multiobjective service replicas placement strategy for fog computing. *IEEE Trans. Cybern.* 51, 11 (2020), 5595–5608.

[53] Mohammad Mainul Islam, Fahimeh Ramezani, Hai Yan Lu, and Mohsen Naderpour. 2023. Optimal Placement of Applications in the Fog Environment: A Systematic Literature Review. *J. Parallel and Distrib. Comput.* 174 (2023), 46–69.

[54] K. Iyeswariya and R. Muthubharathi. 2020. Investigation of edge-fog layer for accuracy and power consumption. In *Proceedings of the 6th International Conference on Advanced Computing and Communication Systems (ICACCS'20).* IEEE, 1252–1255.

[55] Bushra Jamil, Humaira Ijaz, Mohammad Shojafar, Kashif Munir, and Rajkumar Buyya. 2022. Resource allocation and task scheduling in fog computing and internet of everything environments: A taxonomy, review, and future directions. *ACM Comput. Surv.* 54, 11s (2022), 1–38.

[56] Muhammad Aslam Jarwar, Sajjad Ali, Muhammad Golam Kibria, Sunil Kumar, and Ilyoung Chong. 2017. Exploiting interoperable microservices in web objects enabled Internet of Things. In *Proceedings of the 9th International Conference on Ubiquitous and Future Networks (ICUFN'17).* IEEE, 49–54.

[57] Christina Terese Joseph and K. Chandrasekaran. 2019. Straddling the crevasse: A review of microservice software architecture foundations and recent advancements. *Softw.: Pract. Exper.* 49, 10 (2019), 1448–1484.

[58] Akashdeep Kaur, Rajesh Kumar, and Sharad Saxena. 2020. Osmotic computing and related challenges: A survey. In *Proceedings of the 6th International Conference on Parallel, Distributed and Grid Computing (PDGC'20).* IEEE, 378–383.

[59] Kiranpreet Kaur, Fabrice Guillemin, Veronica Quintuna Rodriguez, and Francoise Sailhan. 2022. Latency and network aware placement for cloud-native 5G/6G services. In *Proceedings of the IEEE 19th Annual Consumer Communications & Networking Conference (CCNC'22).* IEEE, 114–119.

[60] Kiranpreet Kaur, Fabrice Guillemin, and Francoise Sailhan. 2022. Container placement and migration strategies for cloud, fog, and edge data centers: A survey. *Int. J. Netw. Manag.* 32, 6 (2022), e2212.

[61] Chao Lei and Hongjun Dai. 2020. A heuristic services binding algorithm to improve fault-tolerance in microservice based edge computing architecture. In *Proceedings of the IEEE World Congress on Services (SERVICES'20).* IEEE, 83–88.

[62] Isaac Lera, Carlos Guerrero, and Carlos Juiz. 2018. Availability-aware service placement policy in fog computing based on graph partitions. *IEEE Internet Things J.* 6, 2 (2018), 3641–3651.

[63] Isaac Lera, Carlos Guerrero, and Carlos Juiz. 2019. YAFS: A simulator for IoT scenarios in fog computing. *IEEE Access* 7 (2019), 91745–91758.

[64] Haiyan Li, Bing Tang, Wei Xu, Feiyan Guo, and Xiaoyuan Zhang. 2022. Application deployment in mobile edge computing environment based on microservice chain. In *Proceedings of the IEEE 25th International Conference on Computer Supported Cooperative Work in Design (CSCWD'22)*. IEEE, 531–536.

[65] Jie Lin, Wei Yu, Nan Zhang, Xinyu Yang, Hanlin Zhang, and Wei Zhao. 2017. A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications. *IEEE Internet Things J.* 4, 5 (2017), 1125–1142.

[66] Marko Luksa. 2017. *Kubernetes in Action*. Simon and Schuster.

[67] Wenkai Lv, Quan Wang, Pengfei Yang, Yunqing Ding, Bijie Yi, Zhenyi Wang, and Chengmin Lin. 2022. Microservice deployment in edge computing based on deep q learning. *IEEE Trans. Parallel Distrib. Syst.* 33, 11 (2022), 2968–2978.

[68] Redowan Mahmud, Ramamohanarao Kotagiri, and Rajkumar Buyya. 2018. Fog computing: A taxonomy, survey and future directions. In *Internet of Everything*. Springer, 103–130.

[69] Redowan Mahmud, Samodha Pallewatta, Mohammad Goudarzi, and Rajkumar Buyya. 2022. Ifogsim2: An extended ifogsim simulator for mobility, clustering, and microservice management in edge and fog computing environments. *Journal of Systems and Software* 190 (2022), 111351.

[70] Redowan Mahmud, Kotagiri Ramamohanarao, and Rajkumar Buyya. 2020. Application management in fog computing environments: A taxonomy, review and future directions. *ACM Comput. Surv.* 53, 4 (2020), 1–43.

[71] Hadi Tabatabaee Malazi, Saqib Rasool Chaudhry, Aqeel Kazmi, Andrei Palade, Christian Cabrera, Gary White, and Siobhán Clarke. 2022. Dynamic service placement in multi-access edge computing: A systematic literature review. *IEEE Access* 10 (2022), 32639–32688.

[72] Mir Gholamreza Mortazavi, Mirsaeid Hosseini Shirvani, and Arash Dana. 2022. A discrete cuckoo search algorithm for reliability-aware energy-efficient IoT applications multi-service deployment in fog environment. In *Proceedings of the International Conference on Electrical, Computer and Energy Technologies (ICECET'22)*. IEEE, 1–6.

[73] Ranesh Kumar Naha, Saurabh Garg, Dimitrios Georgakopoulos, Prem Prakash Jayaraman, Longxiang Gao, Yong Xiang, and Rajiv Ranjan. 2018. Fog computing: Survey of trends, architectures, requirements, and research directions. *IEEE Access* 6 (2018), 47980–48009.

[74] Zahra Makki Nayeri, Toktam Ghafarian, and Bahman Javadi. 2021. Application placement in Fog computing with AI approach: Taxonomy and a state of the art survey. *J. Netw. Comput. Applic.* 185 (2021), 103078.

[75] Benazir Neha, Sanjaya Kumar Panda, Pradip Kumar Sahu, Kshira Sagar Sahoo, and Amir H. Gandomi. 2022. A systematic review on osmotic computing. *ACM Trans. Internet Things* 3, 2 (2022), 1–30.

[76] Sam Newman. 2019. *Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith*. O'Reilly Media.

[77] Yipei Niu, Fangming Liu, and Zongpeng Li. 2018. Load balancing across microservices. In *Proceedings of the IEEE Conference on Computer Communications*. IEEE, 198–206.

[78] Roy Oberhauser. 2016. Microflows: Automated planning and enactment of dynamic workflows comprising semantically-annotated microservices. In *Proceedings of the International Symposium on Business Modeling and Software Design*. Springer, 183–199.

[79] Samodha Pallewatta, Vassilis Kostakos, and Rajkumar Buyya. 2019. Microservices-based IoT application placement within heterogeneous and resource constrained fog computing environments. In *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*. 71–81.

[80] Samodha Pallewatta, Vassilis Kostakos, and Rajkumar Buyya. 2022. QoS-aware placement of microservices-based IoT applications in Fog computing environments. *Fut. Gen. Comput. Syst.* 131 (2022), 121–136.

[81] John Paul Martin, A. Kandasamy, and K. Chandrasekaran. 2020. CREW: Cost and reliability aware eagle-whale optimiser for service placement in Fog. *Softw.: Pract. Exper.* 50, 12 (2020), 2337–2360.

[82] Mohammad Imranur Rahman, Sebastiano Panichella, and Davide Taibi. 2019. A curated dataset of microservices-based systems. In *Joint Proceedings of the Inforte Summer School on Software Maintenance and Evolution (SSSME-2019)*. 1–9.

[83] Nitin Rathore, Anand Rajavat, and Margi Patel. 2020. Investigations of microservices architecture in edge computing environment. In *Social Networking and Computational Intelligence*. Springer, 77–84.

[84] Abdul Razzaq. 2020. A systematic review on software architectures for IoT systems and future direction to the adoption of microservices architecture. *SN Comput. Sci.* 1, 6 (2020), 1–30.

[85] Ju Ren, Deyu Zhang, Shiwen He, Yaoxue Zhang, and Tao Li. 2019. A survey on end-edge-cloud orchestrated network computing paradigms: Transparent computing, mobile edge computing, fog computing, and cloudlet. *ACM Comput. Surv.* 52, 6 (2019), 1–36.

[86] Chris Richardson. 2018. *Microservices Patterns*. Manning Publications Company.

[87] Maria Alejandra Rodriguez and Rajkumar Buyya. 2017. A taxonomy and survey on scheduling algorithms for scientific workflows in IaaS cloud computing environments. *Concurr. Computat.: Pract. Exper.* 29, 8 (2017), e4041.

[88] Oscar Ricardo Cuéllar Rodríguez, Claus Pahl, Nabil El Ioini, Hamid R. Barzegar, et al. 2021. Improvement of edge computing workload placement using multi objective particle swarm optimization. In *Proceedings of the 8th International Conference on Internet of Things: Systems, Management and Security (IOTSMS'21)*. IEEE, 1–8.

[89] Tasneem Salah, M. Jamal Zemerly, Chan Yeob Yeun, Mahmoud Al-Qutayri, and Yousof Al-Hammadi. 2016. The evolution of distributed systems towards microservices architecture. In *Proceedings of the 11th International Conference for Internet Technology and Secured Transactions (ICITST'16)*. IEEE, 318–325.

[90] Farah Ait Salaht, Frédéric Desprez, and Adrien Lebre. 2020. An overview of service placement problem in fog and edge computing. *ACM Comput. Surv.* 53, 3 (2020), 1–35.

[91] Amit Samanta and Jianhua Tang. 2020. Dyme: Dynamic microservice scheduling in edge computing enabled IoT. *IEEE Internet Things J.* 7, 7 (2020), 6164–6174.

[92] Cleber Santana, Brenno Alencar, and Cássio Prazeres. 2018. Microservices: A mapping study for internet of things solutions. In *Proceedings of the IEEE 17th International Symposium on Network Computing and Applications (NCA'18)*. IEEE, 1–4.

[93] Dharmendra Shadija, Mo Rezai, and Richard Hill. 2017. Microservices: Granularity vs. performance. In *Proceedings of the 10th International Conference on Utility and Cloud Computing*. 215–220.

[94] Olena Skarlat, Matteo Nardelli, Stefan Schulte, and Schahram Dustdar. 2017. Towards QoS-aware fog service placement. In *Proceedings of the IEEE 1st International Conference on Fog and Edge Computing (ICFEC'17)*. IEEE, 89–96.

[95] Sven Smolka and Zoltán Ádám Mann. 2022. Evaluation of fog application placement algorithms: A survey. *Computing* 104, 6 (2022), 1397–1423.

[96] Bing Tang, Feiyan Guo, Buqing Cao, Mingdong Tang, and Kuanching Li. 2022. Cost-aware deployment of microservices for IoT applications in mobile edge computing environment. *IEEE Trans. Netw. Serv. Manag.* (2022).

[97] Lam Nguyen Tran Thanh, Nguyen Ngoc Phien, Hong Khanh Vo, Hoang Huong Luong, Tuan Dao Anh, Khoi Nguyen Huynh Tuan, Ha Xuan Son, et al. 2021. IoHT-MBA: An internet of healthcare things (IoHT) platform based on microservice and brokerless architecture. *Int. J. Adv. Comput. Sci. Applic.* 12, 7 (2021).

[98] Muhammad Tirmazi, Adam Barker, Nan Deng, Md E. Haque, Zhijing Gene Qin, Steven Hand, Mor Harchol-Balter, and John Wilkes. 2020. Borg: The next generation. In *Proceedings of the 15th European Conference on Computer Systems*. 1–14.

[99] Prateeksha Varshney and Yogesh Simmhan. 2020. Characterizing application scheduling on edge, fog, and cloud computing resources. *Softw.: Pract. Exper.* 50, 5 (2020), 558–595.

[100] Jagannathan Venkatesh, Baris Aksanli, Christine S. Chan, Alper S. Akyürek, and Tajana S. Rosing. 2017. Scalable-application design for the IoT. *IEEE Softw.* 34, 1 (2017), 62–70.

[101] Massimo Villari, Maria Fazio, Schahram Dustdar, Omer Rana, and Rajiv Ranjan. 2016. Osmotic computing: A new paradigm for edge/cloud integration. *IEEE Cloud Comput.* 3, 6 (2016), 76–83.

[102] Hao Wang, Yong Wang, Guanying Liang, Yunfan Gao, Weijian Gao, and Wenping Zhang. 2021. Research on load balancing technology for microservice architecture. In *Proceedings of the MATEC Web of Conferences*, Vol. 336. EDP Sciences, 08002.

[103] Shangguang Wang, Yan Guo, Ning Zhang, Peng Yang, Ao Zhou, and Xuemin Shen. 2019. Delay-aware microservice coordination in mobile edge computing: A reinforcement learning approach. *IEEE Trans. Mob. Comput.* 20, 3 (2019), 939–951.

[104] Muhammad Waseem, Peng Liang, and Mojtaba Shahin. 2020. A systematic mapping study on microservices architecture in DevOps. *J. Syst. Softw.* 170 (2020), 110798.

[105] Hiroki Watanabe, Tomonori Sato, Takao Kondo, and Fumio Teraoka. 2021. AFC: A mechanism for distributed data processing in edge/fog computing. In *Proceedings of the IEEE Global Communications Conference (GLOBECOM'21)*. IEEE, 01–07.

[106] Zhenyu Wen, Tao Lin, Renyu Yang, Shouling Ji, Rajiv Ranjan, Alexander Romanovsky, Changting Lin, and Jie Xu. 2019. GA-Par: Dependable microservice orchestration framework for geo-distributed clouds. *IEEE Trans. Parallel Distrib. Syst.* 31, 1 (2019), 129–143.

[107] Roy Woodhead, Paul Stephenson, and Denise Morrey. 2018. Digital construction: From point solutions to IoT ecosystem. *Autom. Construct.* 93 (2018), 35–46.

[108] Fulong Xu, Zhenyu Yin, Ai Gu, Feiqing Zhang, and Yue Li. 2020. A service redundancy strategy and ant colony optimization algorithm for multiservice fog nodes. In *Proceedings of the IEEE 6th International Conference on Computer and Communications (ICCC'20)*. IEEE, 1567–1572.

[109] Yangchuan Xu, Lulu Chen, Zhihui Lu, Xin Du, Jie Wu, and Patrick C. K. Hung. 2023. An Adaptive Mechanism for Dynamically Collaborative Computing Power and Task Scheduling in Edge Environment. *IEEE Internet of Things Journal* 10, 4 (2023), 3118–3129.

[110] Saqing Yang, Yi Ren, Jianfeng Zhang, Jianbo Guan, and Bao Li. 2021. KubeHICE: Performance-aware container orchestration on heterogeneous-ISA architectures in cloud-edge platforms. In *Proceedings of the IEEE International Conference on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*. IEEE, 81–91.

[111] Ashkan Yousefpour, Caleb Fung, Tam Nguyen, Krishna Kadiyala, Fatemeh Jalali, Amirreza Niakanlahiji, Jian Kong, and Jason P. Jue. 2019. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *J. Syst. Archit.* 98 (2019), 289–330.

[112] Wei Zhang, Quan Chen, Kaihua Fu, Ningxin Zheng, Zhiyi Huang, Jingwen Leng, and Minyi Guo. 2022. Astraea: Towards QoS-aware and resource-efficient multi-stage GPU services. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems.* 570–582.

[113] Hailiang Zhao, Shuiguang Deng, Zijie Liu, Jianwei Yin, and Schahram Dustdar. 2019. Distributed redundant placement for microservice-based applications at the edge. *arXiv preprint arXiv:1911.03600* (2019).

[114] Hailiang Zhao, Shuiguang Deng, Zijie Liu, Jianwei Yin, and Schahram Dustdar. 2022. Distributed redundancy scheduling for microservice-based applications at the edge. *IEEE Transactions on Services Computing* 15, 3 (2022), 1732–1745.

[115] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chao Ji, Wenhai Li, and Dan Ding. 2018. Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study. *IEEE Trans. Softw. Eng.* 47, 2 (2018), 243–260.

[116] Yousaf Bin Zikria, Rashid Ali, Muhammad Khalil Afzal, and Sung Won Kim. 2021. Next-generation internet of things (IoT): Opportunities, challenges, and solutions. *Sensors* 21, 4 (2021), 1174.