Scheduling IoT Applications in Edge and Fog Computing Environments: A Taxonomy and Future Directions

MOHAMMAD GOUDARZI, MARIMUTHU PALANISWAMI, and RAJKUMAR BUYYA,
The University of Melbourne, Australia

Fog computing, as a distributed paradigm, offers cloud-like services at the edge of the network with low latency and high-access bandwidth to support a diverse range of IoT application scenarios. To fully utilize the potential of this computing paradigm, scalable, adaptive, and accurate scheduling mechanisms and algorithms are required to efficiently capture the dynamics and requirements of users, IoT applications, environmental properties, and optimization targets. This paper presents a taxonomy of recent literature on scheduling IoT applications in Fog computing. Based on our new classification schemes, current works in the literature are analyzed, research gaps of each category are identified, and respective future directions are described.

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

Additional Key Words and Phrases: Fog computing, Internet of Things, Scheduling Taxonomy, Application Structure, Environmental Architecture, Optimization Characteristics, Performance Evaluation

ACM Reference Format:
Mohammad Goudarzi, Marimuthu Palaniswami, and Rajkumar Buyya. 2022. Scheduling IoT Applications in Edge and Fog Computing Environments: A Taxonomy and Future Directions. *ACM Comput. Surv.* 1, 1, Article 1 (January 2022), 43 pages. https://doi.org/nn.nnnn/nnnnnnn.nnnnnnn

1 INTRODUCTION

The Internet of Things (IoT) paradigm has become an integral part of our daily life, thanks to the continuous advancements of hardware and software technologies and ubiquitous access to the Internet. The IoT concept spans a diverse range of application areas such as smart city, industry, transportation, smart home, entertainment, and healthcare, in which context-aware entities (e.g., sensors) can communicate together without any temporal or spatial constraints [41, 52]. Thus, it has shaped a new interaction model among different real-world entities, bringing forward new challenges and opportunities. According to Business Insider [1] and International Data Corporation (IDC) [2], 41 Billion active IoT devices will be connected to the Internet by 2027, generating more than 73 Zettabytes of data. The real power of IoT resides in collecting and analyzing the data circulating in the environment [63], while the majority of IoT devices are equipped with a constrained battery, computing, storage, and communication units, preventing the efficient execution of IoT applications and data analysis on time. Thus, data should be forwarded to surrogate servers for processing and storage. The processing, storage, and transmission of this gigantic amount of IoT data require special considerations while considering a diverse range of IoT applications.

Authors' addresses: M. Goudarzi, M. Palaniswami, and R. Buyya, The Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne, Australia, mgoudarzi@student.unimelb.edu.au.

0360-0300/2022/1-ART1 $15.00
https://doi.org/nn.nnnn/nnnnnnn.nnnnnnn

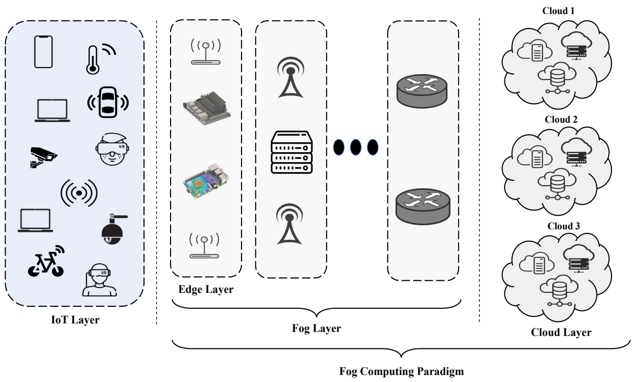

Fig. 1. An illustration of Fog computing environments

1.1 Edge and Fog Computing Paradigms

Cloud computing is one of the main enablers of IoT that offers on-demand services to process, analyze, and store the data generated from IoT devices in a simplified, scalable, and affordable manner [36, 63, 106]. However, the current cloud infrastructure cannot solely satisfy the requirements of a wide range of IoT applications. First, Cloud Data Centers (CDCs) are located at a multi-hop distance from IoT devices, incurring high access latency and data transmission between IoT devices and CDCs. Thus, it poses an important barrier to the efficient service delivery of real-time and latency-sensitive IoT applications. Besides, the service startup time of IoT applications can be negatively affected due to the extended transmission time required for sending commands to distant Cloud Servers (CSs). Also, the extended transmission period and higher latency lead to higher energy consumption for battery-constrained IoT devices. Second, when myriad IoT devices initiate data-driven interactions with applications deployed on remote CSs, it incurs substantial loads on the network and may lead to severe congestion. Third, it increases the computational overhead on CDCs and may reduce their computing efficiency [76]. Fourth, the transmission of sensitive raw data over the Internet is not feasible for some IoT applications due to privacy and security concerns [87, 126]. To address these limitations, Edge and Fog computing paradigms have emerged, bringing Cloud-like services to the proximity of IoT devices [15, 23].

In the Fog computing paradigm, a large number of geo-distributed and heterogeneous Fog Servers (FSs) are located in an intermediate layer between CSs and IoT devices [37, 152]. Distributed FSs (e.g., Raspberry Pis (Rpi), Nvidia Jetson platform, small-cell base stations, nano servers, femtocells, regional servers, core routers, and switches) offer heterogeneous computational and storage resources for IoT devices running various applications, as depicted in Fig. 1. Since FSs are located in the proximity of IoT devices, compared to CDCs, they can offer Cloud-like services with less latency, which effectively address the requirements of real-time and latency-sensitive IoT applications [150]. Moreover, FSs can be accessed with higher bandwidth (i.e., data rate), reducing the required transmission time. Furthermore, Fog computing can help reduce the energy consumption of IoT devices, which is an important parameter, especially for battery-constrained IoT devices. Also, it conserves network bandwidth that decreases the scope of network congestion [85, 107]. Besides, Fog computing helps finer computational load distribution, reducing the massive load on CDCs. Finally, Fog computing enables on-premises pre-processing/processing and storage of raw data, which minimizes the requirement of transmitting raw data to distant servers. In this way, we could mitigate the risks of security breaches and preserve data privacy compared to utilizing raw data.

Compared to CSs, FSs usually have limited resources (e.g., CPU, RAM) while they can be accessed more efficiently. Thus, Fog computing does not compete with Cloud computing, but they complement each other to satisfy diverse requirements of heterogeneous IoT applications. In our view, Edge computing harnesses only distributed Edge resources at the closest layer to IoT devices, while Fog computing harnesses distributed resources located in different layers and also Cloud resources (although some works use these terms interchangeably [75, 113]), as shown in Fig. 1.

1.2 Scheduling IoT Applications in Fog Computing Environments

Fog computing paradigm provides a scalable solution for integrating a diverse range of hardware and software technologies to offer a wide variety of services for end-users. Fog computing environment is highly heterogeneous in terms of end-users' devices, IoT applications, infrastructures, communication protocols, and deployed frameworks. Hence, the smooth execution of IoT applications in this highly heterogeneous computing environment depends on a large number of contributing parameters, making the efficient scheduling of IoT applications an important and yet a challenging problem in Fog computing environments. To effectively utilize the potential of Fog computing paradigm, these challenges should be thoroughly identified.

1.2.1 Challenges of Scheduling IoT Applications. The important challenges of scheduling IoT applications are listed below:

- **Challenges related to IoT devices:** The IoT contains a large variety of devices with heterogeneous resource capabilities such as CPU, RAM, storage, and networking [63]. While IoT devices can be used for the implementation of an unprecedented number of innovative services, their resources are insufficient to host such applications in most scenarios [107]. Besides, a large category of IoT devices are battery-powered and the direct execution of IoT applications on these devices negatively affects their lifetime. Also, many IoT application scenarios consider moving IoT devices (e.g., real-time patient monitoring systems, drones [77]), which further intensify the problem of energy consumption of IoT devices due to increased interference and communication overhead [37].
- **Challenges related to IoT applications:** The number of IoT applications is rapidly increasing due to recent advancements in technology, emerging business models, and end-users' expectations. The design factors of IoT applications (e.g., granularity level, workload type, and interaction model) heavily depend on application scenario and targeted end-users, necessitating a specific amount of resources for smooth execution and different Quality of Service (QoS) requirements. To illustrate, even in one IoT application domain such as healthcare, there are significantly latency-sensitive applications (e.g., sleep apnea analysis [130]) while there are other applications that are more computation-intensive (e.g., radiography [151]). However, the number of computing and communication resources is usually limited. Besides, when several applications are simultaneously requested by the same or different users, the admission control for arriving concurrent requests should be considered as well, making the design factors of an ideal application scheduler even more complex.
- **Challenges related to Edge, Fog, and Cloud resources:** In Fog computing environments, heterogeneous FSs are situated between IoT devices and CDCs through several layers. Usually, it is assumed that FSs are resource-constrained compared to CSs, where FSs at the bottom-most layer (i.e., Edge) have the least amount of resources while providing better access latency and communication bandwidth. Hence, several resource-specific parameters should be considered for scheduling even one IoT application. Besides, considering the ever-increasing number of IoT applications and their diverse resource requirements, some applications cannot be executed on one FS. Thus, FSs require cooperation and resource-sharing with other FSs or

CSs for the execution of IoT applications, making the scheduling problem even more complex. However, resource-sharing among FSs is less resilient than CSs due to spatial constraints and high heterogeneity in deployed operating systems, standards, and protocols, just to mention a few [67, 76, 79]. In addition, FSs are more exposed to end-users which makes them potentially less-secured compared to CSs [126]. Besides, the geo-distribution of resources and the IoT data hosted and shared on different servers may also impose privacy implications. As many IoT users may share personal information in Fog computing environments, adversaries can gain access to this shared information [121].

- **Challenges related to optimizing parameters** Optimizing the performance of IoT applications running in Fog computing environments depends on numerous parameters such as the main goal of each IoT application, the capabilities of IoT devices, servers properties, networking characteristics, and the imposed constraints. Optimizing the performance of even one IoT application in such a heterogeneous environment with numerous contributing parameters is complex, while multiple IoT applications with different parameters and goals further complicate the problem.
- **Challenges related to decision engines:** Decision engines are responsible to collect all contextual information and schedule IoT applications. Based on the context of IoT applications and environmental parameters, these decision engines may use different optimization modeling [68]. Besides, there are several placement techniques to solve these optimization problems. However, considering the types of IoT application scenarios and the number and types of contributing parameters, different placement techniques lead to completely different performance gain [39, 129]. For example, some placement techniques result in high-accuracy decisions while their decision time takes a long time. However, some other techniques find acceptable solutions with shorter decision time. Moreover, the decision engines can be equipped with several advanced features such as mobility support and failure recovery, enabling them to work in more complex environments.
- **Challenges related to real-world performance evaluation:** The lack of global Fog service providers offering infrastructure on pay-as-you-go models like commercial Cloud platforms such as Microsoft Azure and Amazon Web Services (AWS) pushes researchers to set up small-scale Edge/Fog computing environments [77]. The real-world performance evaluation of IoT applications and different placement techniques in Fog computing is not as straightforward as Cloud computing since the management of distributed FSs incurs extra effort and cost, specifically in large-scale scenarios. Besides, the modification and tuning of system parameters during the experiments are time-consuming. Hence, while real-world implementations are the most accurate approach for the performance evaluation of the systems, it is not always feasible, specifically in large-scale scenarios.

1.2.2 Motivation of Research. Numerous techniques for scheduling IoT applications in Fog computing environments have been developed to address the above-mentioned challenges. Several works focused on the structure of IoT applications and how these parameters affect the scheduling [29, 79, 101] while some other techniques mainly focus on environmental parameters of Fog computing, such as the effect of hierarchical Fog layers on the scheduling of IoT applications [38, 62]. Besides, several techniques focus on defining specific optimization models to formulate the effect of different parameters such as FSs' computing resources, networking protocols, and IoT devices characteristics, just to mention a few [68]. Moreover, several works have proposed different placement techniques to find an acceptable solution for the optimization problem [5, 51, 78] while some other techniques consider mobility management [26, 38, 131] and failure recovery [8, 40]. All these perspectives directly affect the scheduling problem, especially, when designing decision

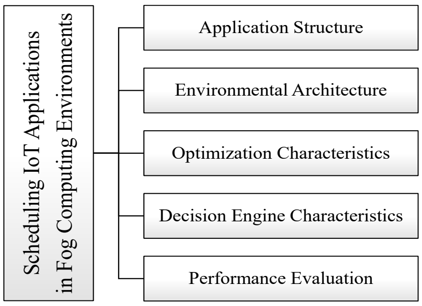
Scheduling IoT Applications in Fog Computing Environments
Application Structure
Environmental Architecture
Optimization Characteristics
Decision Engine Characteristics
Performance Evaluation

Fig. 2. Different perspectives of scheduling IoT applications in Fog computing

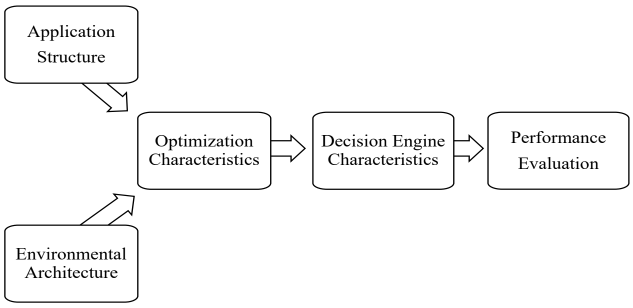
Application Structure
Environmental Architecture
Optimization Characteristics
Decision Engine Characteristics
Performance Evaluation

Fig. 3. The relation among different identified perspectives

engines. These perspectives should be simultaneously considered when studying and evaluating each proposal. However, only a few works in the literature have identified the scheduling challenges that directly affect the designing and evaluation of decision engines in Fog computing environments and accordingly categorized proposed works in the literature. Thus, we identify five important perspectives regarding scheduling IoT applications in Fog computing environments, as shown in Fig. 2, namely application structure, environmental architecture, optimization modeling, decision engines' characteristics, and performance evaluation.

Fig. 3 depicts the relationships among identified perspectives. The features of application structure and environmental architecture help define the optimization characteristic and formulate the problem. Then, an efficient decision engine is required to effectively solve the optimization problem. Besides, the performance of the decision engine should be monitored and evaluated based on the main goal of optimization for the target applications and environment. Considering each perspective, we present a taxonomy and review the existing proposals. Finally, based on the studied works, we identify the research gaps in each perspective and discuss possible solutions. The main contributions of this work are:

- We review the recent literature on scheduling IoT applications in Fog computing from application structure, environmental architecture, optimization modeling, decision engine characteristics, and performance evaluation perspectives and propose separate taxonomies.
- We identify research gaps of scheduling IoT applications in Fog computing considering each perspective.
- We present several guidelines for designing a scheduling technique in Fog computing paradigm.

- We identify and discuss several future directions to help researchers advance Fog computing paradigm.

1.3 Paper Organization

The rest of the paper is organized as follows. The existing related surveys and taxonomies on scheduling IoT applications in Fog computing environments are studied and compared with ours in Section 2. Section 3 presents a taxonomy and overview of the IoT applications' structure. Section 4 introduces a taxonomy on environmental properties of resources in Fog computing environments and studies the existing works accordingly. In Section 5, a taxonomy of optimization characteristics of problems in Fog computing environments is introduced. Section 6 identifies the important aspects of decision engines and presents a taxonomy of decision engines for scheduling IoT applications. Section 7 demonstrates the approaches and metrics used for the evaluation of scheduling strategies in Fog computing. Section 8 presents a guideline for designing a scheduling technique. According to identified research gaps, Section 9 provides several future research directions. Finally, Section 10 concludes this survey.

2 RELATED SURVEYS

In the context of Fog computing, surveys targeted different aspects of Fog computing, such as security [112, 121, 126, 154], smart cities [105], live migration techniques [99], existing software and hardware [107], deep learning applications [139], healthcare [66], and general surveys studied the Fog computing paradigm, its scope, architectures, and recent trends [10, 44, 52, 91–93, 96, 122]. Also, some surveys mainly discussed resource management, application management, and scheduling in the context of Fog computing, such as [3, 34, 60, 68, 79, 86, 113, 118, 152], that we discuss and compare them with ours.

Aazam et al. [3] reviewed enabling technologies and research opportunities in Fog computing environments alongside studying computation offloading techniques in different domains such as Fog, Cloud, and IoT. Hong et al. [48] and Ghobaei-Arani [34] studied resource management approaches in Fog computing environments and discussed the main challenges for resource management. Yousefpour et al. [152] discussed the main features of the Fog computing paradigm and compared it with other related computing paradigms such as Edge and Cloud computing. Besides, it studied the foundations, frameworks, resource management, software, and tools proposed in Fog computing. Mahmud et al.[79] mainly discussed the application management and maintenance in Fog computing and proposed a taxonomy accordingly. Salaht et al. [113] presented a survey of current research conducted on service placement problems in Fog Computing and categorized these techniques. Shakarami et al. [118] studied machine learning-based computation offloading approaches while Adhikari et al. presented the type and applications of nature-inspired algorithms in the Edge computing paradigm. Martinez et al. [86] mainly focused on designing and evaluating Fog computing systems and frameworks. Lin et al. [68] and Sonkoly et al. [123] mainly studied and categorized different approaches for modeling the resources and communication types for computation offloading in Edge computing. Finally, Islam et al. [60] proposed a taxonomy for context-aware scheduling in Fog computing and surveyed the related techniques in terms of contextual information such as user and networking characteristics.

Table 1 summarizes the characteristics of related surveys and provides a qualitative comparison with our work. The proper scheduling of IoT applications in Fog computing environments can be viewed from different perspectives, such as application structure, environmental architecture, optimization modeling, and the features of decision engines. Besides, the performance of scheduling techniques should be continuously evaluated to offer the best performance. As depicted in Table 1, the existing surveys barely study and provide comprehensive taxonomy for the above-mentioned

Table 1. Related surveys on scheduling in Fog computing

Survey	Application Structure		Environmental Architecture		Optimization Characteristics		Decision Engine		Performance Evaluation		Conceptualize Scheduling Framework	Research Gap (in Years)
	Taxonomy	Research Gaps	Taxonomy	Research Gaps	Taxonomy	Research Gaps	Taxonomy	Research Gaps	Taxonomy	Research Gaps		
[3]	○	◐	○	○	○	○	○	○	○	○	●	4
[48]	○	○	◐	◐	◐	○	◐	○	○	○	○	3.5
[152]	○	○	◐	◐	○	○	○	◐	○	○	○	3
[34]	○	○	○	○	○	○	◐	◐	○	○	○	2.5
[113]	○	○	○	○	◐	●	◐	◐	●	◐	○	2
[79]	●	●	○	◐	○	○	◐	◐	○	○	●	1.5
[118]	○	○	○	○	○	○	◐	◐	○	○	○	1.5
[86]	○	○	○	○	○	○	○	◐	◐	◐	○	1.5
[68]	○	○	◐	○	●	◐	○	◐	○	○	○	1.5
[60]	○	◐	◐	○	○	○	○	◐	○	○	○	1
[123]	◐	○	○	○	◐	◐	○	○	○	○	○	0.5
[7]	○	○	○	○	○	◐	○	◐	○	○	○	0.5
This Survey	●	●	●	●	●	●	●	●	●	●	●	Current

●: Full Cover, ◐: Partial Cover, ○ : Does Not Cover

perspectives. In this work, we identify the main parameters of each perspective and present a taxonomy accordingly. Moreover, we identify related research gaps and provide future directions to improve the Fog computing paradigm.

3 APPLICATION STRUCTURE

The primary goal of Fog computing is to offer efficient and high-quality service to users with heterogeneous applications and requirements. Hence, service providers require a comprehensive understanding of each IoT application structure (e.g., workload model and latency requirements) to better capture its complexities, perform efficient scheduling and resource management, and offer high-quality service to the users. Also, when designing the architecture of each IoT application, dynamics, constraints, and complexities of resources in Fog computing should be carefully considered to exploit the potential of this paradigm. Fig. 4 presents a taxonomy and main elements of IoT application structure, described below.

3.1 Architectural Design

The logic of IoT applications can be implemented in different ways. To illustrate, operations of a Video Optical Character Recognition (VideoOCR) such as capturing frames, similarity check, and text extraction can be implemented as a single encapsulated program or as a set of interdependent components [24]. Hence, according to the granularity level of applications, their distribution, and coupling intensity, the architectural design of applications can be classified into four types:

3.1.1 Monolithic. It encapsulates the complete logic of an application as a single component or program. The parallel execution of these applications can be obtained using multi processing approaches [79]. In the context of Fog computing, several works such as [13, 20, 84, 89] have considered monolithic applications.

3.1.2 Independent. These applications require the execution of a set of independent tasks or components for the complete execution of the application. The constituent parts of these applications can be simultaneously executed on different FSs or CSs. Several works such as [9, 46, 55, 104] discuss applications with independent components or tasks in the Fog literature.

3.1.3 Modular. Each modular application is composed of a set of dependent tasks or components. While constituent parts of each application can be distributed over several FSs or CSs for parallel execution, there are some constraints for the execution of tasks based on their data-flow dependency model. Several works in the literature such as [39, 70, 72, 103] discuss modular applications.

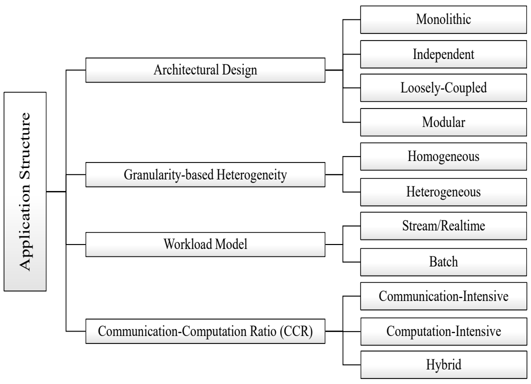

Fig. 4. Application structure taxonomy

3.1.4 Loosely-coupled. Components of loosely-coupled applications (i.e., microservices) can be distributed over several CSs or FSs. Besides, due to service-level isolation, components of applications can be shared among different applications, providing high application extendability. Several works such as [24, 25, 38, 137] have considered loosely-coupled applications.

3.2 Granularity-based Heterogeneity

Tasks within an IoT application have different properties such as computation size, input size, output size, and deadline. These features affect the scheduling complexity, where identifying the dynamics of applications with heterogeneous task properties requires further effort. Accordingly, we categorize IoT applications based on their granularity-level specifications to 1) **heterogeneous** such as [47, 70] or 2) **homogeneous** such as [55, 153].

3.3 Workload Model

The workload model specifies the data architecture of an application, which can be broadly divided into two categories for IoT applications:

3.3.1 Stream/Realtime. In this category, the data should be processed by the servers as soon as it was generated (i.e., real-time), and hence, the data usually require relatively simple transformation or computation. Several works such as [22, 65, 82] discuss stream workload for IoT applications.

3.3.2 Batch. In batch processing, the input data of an application is usually bundled for processing. However, contrary to heavy batch processing models, IoT applications often use micro-batches to provide a near-realtime experience. In the literature, several works such as [19, 45, 138] consider batch workload for the applications.

3.4 Communication-Computation Ratio (CCR)

Each IoT application, regardless of its architecture, contains some amount of input data for transmission and computational load for processing. These properties can significantly affect the scheduling decision to find proper FSs or CSs for an application. The CCR defines whether an application on average is more 1) **computation-intensive** [42, 101, 132] or 2) **communication-intensive** [13, 49, 116]. Besides, some works consider a range of applications to cover both computation-intensive and communication-intensive applications, to which we refer as 3) **hybrid** [16, 40].

3.5 Discussion

In this section, we discuss the effects of identified application structure's elements on the decision engine and describe the lessons that we have learned. Besides, we identify several research gaps accordingly. Table 2 summarizes the characteristics related to IoT application structure in Fog computing.

3.5.1 ***Effects on the decision engine.*** The application structure properties affect the decision engine in various aspects, as briefly described below.

1. Architectural design: It defines the number of tasks/modules and their respective dependency within a single application. Hence, as the number of tasks/modules per application increases, the problem space significantly increases. Considering an application with n number of tasks and m possible candidate configuration per task, the Time Complexity (TC) of finding the optimal solution is $O(m^n)$. Besides, the dependency of tasks within an application imposes hard constraints on the problem, which further increases the complexity. Thus, finding the optimal solution for the scheduling of applications becomes very time-consuming, and the design of an efficient placement technique to serve applications in a timely manner remain an important yet challenging problem.

2. Granularity-based heterogeneity: It shows the corresponding properties of each task/module within an application and plays a principal role in identifying the dynamics of applications. One of the most important features of decision engines is their adaptability and their capability to extract the complex dynamics of applications so that the decision engine can receive diverse types of applications' requests. Since applications with heterogeneous granularity-based properties have higher dynamics' complexity, the decision engines designed for this application category should support high adaptability.

3. Workload model and CCR: These elements provide insightful information regarding the input data architecture of the application and its behavior in the runtime. Accordingly, the decision engine may define different priority queues for incoming requests based on their workload model and CCR to provide higher QoS for the users. For example, applications with real-time workload types and communication-intensive CCR may have higher priority for the placement on servers closer to the IoT devices than computation-intensive applications that are not real-time.

3.5.2 ***Lessons learned.*** Our findings regarding the IoT application structure in the surveyed works are briefly described in what follows:

1. Almost 70% of the surveyed works have overlooked studying the dependency model of tasks within an application and selected either the independent or monolithic design. The rest of the works consider dependency among tasks of an application in different models (i.e., sequential, parallel, or hybrid dependency). Moreover, only about 10% of the studied works consider microservices in their application design.

2. The most realistic assumption for the granular properties of each task/module is heterogeneous (i.e., heterogeneous input size, output size, and computation size). Almost 85% of the studied works consider heterogeneous properties for each task/module, while around 15% of the works consider the homogeneous properties for the tasks/modules.

3. The workload model and CCR in each proposal depend on the targeted application scenarios. Almost 55% of the works did not study the CCR, or the required information to obtain the CCR (i.e., computation size of tasks, average data size) was not mentioned. Among the rest of the works, computation-intensive, communication-intensive, and hybrid CCR form roughly 25%, 15%, and 5% of proposals respectively.

3.5.3 ***Research Gaps.*** We have identified several open issues for further investigation, that are discussed below:

1. According to Alibaba's data of 4 million applications, more than 75% of the applications consist of dependent tasks [155]. However, only around 30% of the recent works surveyed in this study consider applications with dependent tasks (i.e., modular or loosely-coupled), showing further investigation is required to identify the dynamics of these types of complex applications.

2. Although the microservice-based applications can significantly benefit the IoT scenarios, only a few works such as [38, 137] have studied the scheduling and migration of microservices in Edge/Fog computing environments. So, further investigation is required to study the behavior of microservice-based applications with different resource management techniques.

3. Modular or loosely-coupled IoT applications can be distributed over different FSs or CSs for parallel execution. However, several works such as [78] statically assign components of an application on pre-defined FSs or CSs and only schedule one or two remaining components. Hence, the best placement configuration of applications based on the current dynamics of the system cannot be investigated, leading to diminished performance gain.

4. When the number of IoT applications increases, there is a high probability that application requests with different workload models are submitted to the system. However, none of the studied works in the literature consider applications with different workload models and how they may mutually affect each other in terms of performance.

5. Due to the high heterogeneity of IoT applications in Fog, applications with diverse CCR may be submitted for processing, requiring special consideration such as networking and prioritization. Although there are only a few recent works such as [22, 39] that consider hybrid CCR, most of the recent works target one of the computation-intensive or communication-intensive applications.

Table 2. Summary of existing works considering application structure taxonomy

Ref	Application Structure				Ref	Application Structure			
	Design	Granular Hetero	Workload Model	CCR		Design	Granular Hetero	Workload Model	CCR
[9]	Independent	Hetero	Batch	Comp	[81]	Modular	Hetero	Stream	ND
[13]	Monolithic	Hetero	Batch	Comm	[27]	Monolithic	Hetero	Batch	Comp
[18]	Independent	Hetero	Batch	Comm	[55]	Independent	Homo	Batch	ND
[25]	Loosely-coupled	Hetero	Batch	ND	[89]	Monolithic	Homo	ND	ND
[39]	Modular	Hetero	Batch	Hybrid	[20]	Independent	Hetero	Batch	Comm
[40]	Modular	Hetero	Batch	Hybrid	[56]	Independent	Hetero	Batch	ND
[84]	Monolithic	Homo	Batch	ND	[73]	Independent	Hetero	Batch	Comm
[70]	Modular	Hetero	Batch	Comp	[144]	Independent	Hetero	ND	ND
[129]	Independent	Hetero	Stream	Comp	[31]	Independent	Hetero	Batch	ND
[156]	Independent	Hetero	Stream	ND	[72]	modular	Hetero	Stream	ND
[47]	Monolithic	Hetero	Batch	Comm	[38]	Loosely-coupled	Hetero	Stream	Comp
[54]	Monolithic	ND	ND	ND	[141]	Independent	Hetero	Batch	Comp
[147]	Monolithic	Homo	Batch	Comp	[100]	Monolithic	Hetero	ND	ND
[137]	Loosely-coupled	Hetero	Stream	Comm	[119]	Loosely-coupled	Hetero	Stream	ND
[140]	Independent	Hetero	Stream	Comm	[133]	Monolithic	Homo	Batch	ND
[97]	Modular	Hetero	Batch	Comp	[71]	Monolithic	Homo	Batch	Comp
[125]	Independent	Hetero	Stream	Comm	[148]	Independent	Hetero	Batch	ND
[127]	ND	ND	Batch	ND	[109]	Modular	Hetero	Batch	ND
[32]	Independent	Hetero	Stream	ND	[157]	Independent	Hetero	Batch	ND
[146]	Independent	Hetero	ND	Comp	[153]	Monolithic	Homo	ND	ND
[136]	Monolithic	Homo	ND	Comp	[111]	Independent	Hetero	Batch	ND

[28]	Modular	Hetero	Batch	ND	[120]	Monolithic	Hetero	Batch	Comp
[132]	Loosely-coupled	Hetero	Stream	Comp	[143]	Independent	Hetero	Batch	ND
[138]	Independent	Hetero	Batch	ND	[83]	Independent	Hetero	ND	ND
[42]	Independent	Hetero	Batch	Comp	[117]	Loosely-coupled	Hetero	ND	ND
[45]	Independent	Hetero	Batch	ND	[90]	Modular	Hetero	Batch	ND
[149]	Loosely-coupled	Hetero	Stream	Comp	[14]	Independent	ND	Stream	ND
[53]	Independent	Hetero	Stream	Comp	[58]	Independent	Hetero	Stream	ND
[64]	Loosely-coupled	Hetero	Batch	Comp	[114]	Loosely-coupled	Hetero	Batch	ND
[128]	Monolithic	Hetero	Stream	Comp	[6]	Independent	Hetero	Batch	ND
[33]	Independent	Hetero	Batch	Comp	[16]	Modular	Hetero	Batch	Hybrid
[19]	Independent	Hetero	Batch	ND	[22]	Independent	Hetero	Stream	Hybrid
[11]	Monolithic	Homo	Batch	ND	[4]	Independent	Hetero	Batch	Comp
[12]	Independent	Hetero	ND	ND	[50]	Independent	Hetero	Batch	ND
[21]	Monolithic	Homo	Batch	ND	[59]	Modular	Hetero	Batch	ND
[74]	Monolithic	Homo	Batch	Comp	[61]	Independent	Hetero	Batch	ND
[30]	Loosely-coupled	Hetero	ND	Comm	[65]	Independent	Hetero	Stream	ND
[155]	Modular	Hetero	Batch	Comm	[69]	Modular	Hetero	Batch	ND
[135]	Modular	Hetero	Batch	Comp	[82]	Independent	Hetero	Stream	ND
[110]	Independent	Hetero	Stream	ND	[95]	Independent	Hetero	Stream	ND
[88]	Independent	Hetero	Stream	ND	[101]	Loosely-coupled	Hetero	Stream	Comp
[142]	Modular	Hetero	Batch	Comm	[98]	Independent	Hetero	Stream	ND
[24]	Loosely-coupled	Hetero	Stream	ND	[103]	Modular	Hetero	Stream	ND
[49]	Monolithic	Homo	Batch	Comm	[104]	Independent	Hetero	Batch	ND
[145]	Modular	Homo	Batch	ND	[108]	Loosely-coupled	Hetero	Stream	ND
[57]	Independent	Hetero	ND	Comp	[115]	Independent	Hetero	Batch	ND
[80]	Independent	Hetero	Stream	ND	[116]	Independent	Hetero	Batch	Comm
[94]	Independent	Hetero	Batch	ND	[134]	Modular	Hetero	Batch	Comp
[124]	Modular	Hetero	Batch	Hybrid	[46]	Independent	Hetero	Batch	ND
[78]	Modular	Hetero	ND	ND	[102]	Modular	Hetero	Batch	Comp

ND: Not Defined, Hetero: Heterogeneity/heterogeneous, Comp: Computation-Intensive, Comm: Communication-Intensive
Homo: Homogeneous

4 ENVIRONMENTAL ARCHITECTURE

The configuration and properties of IoT devices and resource providers directly affect the complexity and dynamics of scheduling IoT applications. To illustrate, as the number of resource providers increases, heterogeneity in the system also grows as a positive factor, while the complexity of making a decision also increases that may negatively affect the process of making decisions. In this section, we classify the environmental architecture properties, as depicted in Fig. 5, into the following categories:

4.1 Tiering Model

IoT devices and resource providers can be conceptually organized in different tiers based on their proximity to users and resources, described below:

4.1.1 Two-Tier. In this resource organization, IoT devices are situated at the bottom-most layer and resource providers are placed at the edge of the network in the proximity of IoT devices (i.e., Edge computing). Several works use two-tier resource organization such as [55, 69, 84, 125].

4.1.2 Three-Tier. Compared to two-tier model, this model also uses CSs at the highest-most layer to support edge resources (i.e., Fog computing). Several works considered three-tier model in the literature such as [59, 65, 108, 116].

4.1.3 Many-Tier. In many-tier resource organizations, IoT devices and CSs are situated at the bottom-most and highest-most tiers respectively, while FSs are placed in between through several tiers (i.e., hierarchical Fog computing). In the literature, several works have considered many-tier model such as [31, 38, 83, 111].

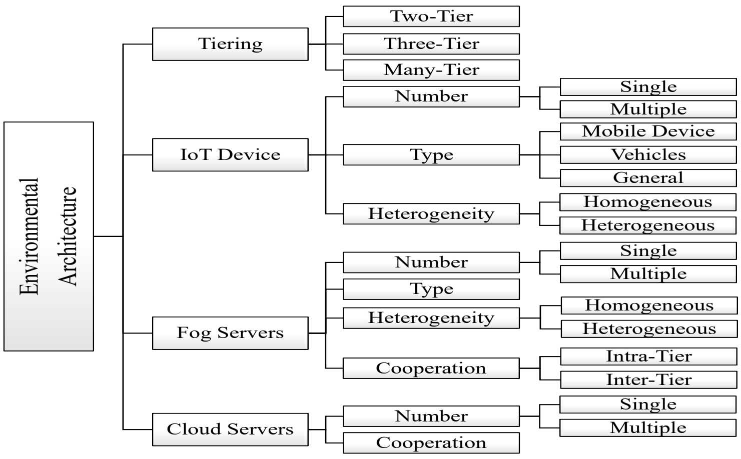

Fig. 5. Environmental architecture taxonomy

4.2 IoT Devices

IoT devices can play two roles; as service requester and/or resource provider. When they act as service requesters, still they can execute a portion of their tasks or components based on their available resources. Moreover, IoT devices can simultaneously play these different roles. We study the properties of IoT devices from the following perspectives:

4.2.1 Number. The higher number of IoT devices (either as service requester or service provider), the higher complexity of the scheduling problem. Some works only consider single IoT device in the environment such as [70, 97, 138] while other works consider multiple IoT devices simultaneously such as [50, 100, 133].

4.2.2 Type. The type of IoT devices help us understand the amount of resources, capabilities, and constraints of these devices. The IoT devices used in the current literature can be broadly classified into three categories, namely 1) **mobile devices (MD)** which are mostly considered as smartphones or tablets [25, 102, 119], 2) **Vehicles** [137, 140, 148], and 3) **General** devices containing a set of IoT devices, ranging from small sensors to drones [39, 84, 120].

4.2.3 Heterogeneity. We also study the resources of IoT devices and their request types, and classify proposals into 1) **heterogeneous** where IoT devices have various resources and different request types and sizes such as [31, 72, 145] or 2) **homogeneous** where the resources of IoT devices are the same or they have the same request type and size such as [70, 138, 149].

4.3 Fog Servers (FSs)

FSs usually act as resource providers for IoT devices. The environmental properties of FSs can be classified based on the following criteria:

4.3.1 Number. Similar to IoT devices, we classify the number of FSs in the environment into 1) **Single** and 2) **Multiple**. The complexity and dynamics of system in surveyed works that have considered only single FS such as [53, 54, 110] is simpler to the works that have considered multiple FSs such as [28, 127, 146].

4.3.2 Type. The type of FSs acting as service provider in the Fog computing environment ranges from IoT devices with additional resources to resource-rich data centers. Several works have considered a specific type of FS and discuss their properties in their works such as 1) **Base Station (BS)** and **Macro-cell Station (MS)** [18, 127, 138], 2) **femtocells** [19, 32, 37], 3) **Rpi** [30] and 4) **access points (AP)** [27, 88]. Moreover, several works consider 5) **general** FSs containing a set of FSs with different types such as [31, 94].

4.3.3 Heterogeneity. We study the FSs' resources and classified works based on their heterogeneity into 1) **heterogeneous** and 2) **homogeneous** accordingly. Many works have considered heterogeneous resources for FSs [9, 39, 46, 102] while some works consider homogeneous resources for FSs [54, 73, 134].

4.3.4 Cooperation. Compared to CSs, each FS has fewer resources and it may not be able to satisfy the requirements of IoT applications. Cooperation among FSs helps augmenting their resources and providing service for demanding IoT applications. We classify proposals based on their cooperation among FSs into 1) **intra-tier** where FSs of same tier collaborate to satisfy users' requests [24, 25, 25, 109] and 2) **inter-tier** where FSs of different layers also collaborate for the execution of IoT one application [38, 39].

4.4 Cloud Servers (CSs)

The environmental properties of CSs can be classified based on the following criteria:

4.4.1 Number. The current literature based on the CSs' number can be divided into 1) **single** and 2) **multiple** categories. Majority of works only consider one CDC as resource provider (either as a central entity with aggregated resources or different number of VMs) to support FSs such as [14, 50, 59, 65]. In real-world environment, different CDCs are available which can provide services with different QoS for multiple applications. Some works such as [35, 39, 40, 132] have considered multiple CDCs with heterogeneous CSs in the literature.

4.4.2 Cooperation. Among the works considered multi CDCs, we study either CSs from different CDCs are configured to collaboratively execute an IoT application or not. In the literature, some works such as [35, 40, 104] have considered collaborative multi CDCs scenarios.

4.5 Discussion

In this section, we discuss the effects of identified environmental architecture's elements on the decision engine and describe the lessons that we have learned. Besides, we identify several research gaps accordingly. Table 3 provides a summary of properties related to environmental architecture in Fog computing.

4.5.1 ***Effects on the decision engine.*** The elements of environmental architecture affect the decision engine in various aspects, as briefly described below.

1. Tiering: It represents the organization of end-users' devices and resources in the computing environment. Considering the properties of resources in different tiers, it helps find the most suitable deployment layer for the decision engine to efficiently serve IoT applications' requests with a wide variety of requirements. For example, to serve real-time IoT applications with low startup time requirements, the most suitable deployment layer in the three-tier model is the lowest-level Fog layer.

2. IoT devices: The number of IoT devices directly relates to the number of incoming requests to decision engines. It affects the admission control of decision engines. The type of IoT devices provides contextual information about the number of resources and intrinsic properties of the IoT devices that are important for the decision engine. For example, the MD type not only states that the IoT device does not have significant computing resources, but also presents that the device has mobility features. Thus, the IoT device type affects the advanced features of the decision engine, such as mobility, and also specifies whether the IoT devices can serve one or several tasks/modules of IoT applications or not.

3. Fog and Cloud servers: The number of available servers directly affects the TC of the scheduling problem. Considering an application with n number of tasks and m possible candidate configuration per task, the TC of finding the optimal solution is $O(m^n)$. Hence, it directly affects the choice of placement technique and scalability feature of the decision engine. As the problem space increases, a suitable decision engine should be selected to solve the scheduling problem. Moreover, the type and heterogeneity of resources provide further contextual information for the decision-making, such as the number of resources, networking characteristics, and resource constraints, just to mention a few.

4.5.2 ***Lessons learned***. Our findings regarding the environmental architecture in the surveyed works are briefly described in what follows:

1. Almost 60% of works consider the three-tier model and many-tier models for the organization of end-users and resources. Not only do these works consider real-time applications, but also some of them assume both real-time and computation-intensive applications, such as [16, 39, 124]. This is mainly because these works use CSs as a backup plan for more computation-intensive applications or when the number of incoming IoT requests increases and the FSs cannot solely manage the incoming requests. Moreover, nearly 40% of surveyed works assume a two-tier model for the organization of end-users and resources. These works mostly assume real-time workload type and communication-intensive applications for the deployment on Edge servers, such as [13, 22, 53, 110].

2. In the surveyed works, almost 90% of the works considered an environment with multiple IoT devices, while 10% of works only focused on a single IoT device. When the number of IoT devices increases, the diversity of IoT applications and heterogeneity of their tasks also increase accordingly. Moreover, the greater number of works assume IoT devices as general devices with sensors, actuators, and diverse application requests. In contrast, some works targeted a specific IoT devices such as mobile devices and vehicles with almost 30% and 10% of proposals, respectively. These proposals studied other contextual information of targeted IoT devices in detail such as mobility [138], energy consumption [140], and networking characteristics [109, 137]. Finally, about 90% of works have studied IoT devices with heterogeneous properties and diverse application request types, which are the closest scenario to real-world computing environments.

3. Regarding Fog resources, almost 90% of the proposals consider multiple FSs in the environment. However, only 40% of the current literature has considered any cooperation model among FSs. There is a high probability that a single FS cannot solely manage the execution of several incoming resources due to its limited resources. Also, sending partial/complete applications' tasks to the Cloud may negatively affect IoT devices' response time and energy consumption, especially for real-time IoT applications. Thus, cooperation among FSs is of paramount importance that can lead to the execution of IoT applications with better performance and QoS. Considering the type of the FSs, about 60% of the studied literature considered general FSs. The rest of the works studied a specific type of FSs and tried to involve their contextual information in the scheduling process of IoT applications, such as networking characteristics [9]. Moreover, some works considered IoT devices can simultaneously play different roles in the computing environments (i.e., service requester and service provider) such as [132, 140, 157].

4. In the current literature, around 60% of the works consider CSs as computing resources in the environment. However, only in 8% of these works multiple Cloud service providers (i.e., multi-Cloud), their communication, and interactions are studied, such as [39, 40, 104, 132].

4.5.3 ***Research Gaps.*** We have identified several open issues for further investigation, as discussed below:

1. Hierarchical Fog computing (i.e., multi-tier) has not been thoroughly considered by researchers. Only a few works (almost 5%) consider the organization of resources in the multi-tier environment, and most have focused on the heterogeneity of resources among different tiers. However, these works have not considered the heterogeneity of communication protocols and latency in multi-tier environments.

2. In the literature, several works have considered abstract CDC as a central unit with huge computing capacity [80], while in reality, CDCs contain several CSs hosting computing instances. Such assumptions affect the computing and communication time in simulation studies.

3. One of the main advantages of Fog computing is providing heterogeneous FSs in IoT devices' vicinity to collaboratively serve applications. However, many works have not considered cooperation among FSs. In this case, due to the limited computing and communication resources of each FS and a large number of IoT requests, the serving FS may become a bottleneck which negatively affects the response time and QoS [37]. Besides, in uncooperative scenarios, the overloaded FS forwards requests to CSs, incurring higher latency. Hence, cooperative Fog computing, associated protocols, and constraints require further investigation for different IoT application scenarios.

Table 3. Summary of existing works considering environmental architecture taxonomy

Ref	Environmental Architecture									
	Tiering	IoT Device			Fog Servers				Cloud Servers	
		Number	Type	Hetero	Number	Type	Hetero	Coop	Number	Coop
[9]	Two-Tier	Multiple	MD	Hetero	Multiple	BS, MS	Hetero	○	○	○
[13]	Two-Tier	Multiple	Vehicle	Hetero	Multiple	RSU	ND	○	○	○
[18]	Three-Tier	Multiple	MD	ND	Multiple	BS	Hetero	Intra	Single	○
[25]	Three-Tier	Multiple	MD	Hetero	Multiple	BS	Hetero	Intra	Single	○
[39]	Many-Tier	Multiple	General	Hetero	Multiple	General	Hetero	Intra, Inter	Multiple	●
[40]	Three-Tier	Multiple	General	Hetero	Multiple	General	Hetero	Intra	Multiple	●
[84]	Two-Tier	Multiple	General	Hetero	Multiple	Cloudet	Hetero	○	○	○
[70]	Three-Tier	Single	General	Homo	Multiple	General	Hetero	Intra	Single	○
[129]	Three-Tier	Multiple	General	Homo	Multiple	General	Hetero	ND	Single	○
[156]	Two-Tier	Multiple	General	ND	Multiple	BS, MS	Hetero	○	○	○
[47]	Three-Tier	Multiple	MD	Hetero	Multiple	BS	Hetero	Intra	Single	○
[54]	Two-Tier	Multiple	MD	Hetero	Single	BS	Homo	○	○	○
[147]	Two-Tier	Multiple	MD	Hetero	Multiple	BS	Hetero	Intra	○	○
[137]	Three-Tier	Multiple	Vehicle	Hetero	Multiple	Hybrid	Hetero	Intra	Single	○
[140]	Three-Tier	Multiple	Vehicle	Hetero	Multiple	BS, Vehicle	Homo	○	Single	○
[97]	Three-Tier	Single	MD	Hetero	Single	General	Homo	○	Single	○
[125]	Two-Tier	Multiple	MD	Hetero	Multiple	General	Hetero	○	○	○
[127]	Three-Tier	Multiple	MD	Hetero	Multiple	BS	Hetero	Intra	Single	○
[32]	Three-Tier	Multiple	MD	Hetero	Multiple	Femto	Hetero	Intra	Single	○
[146]	Three-Tier	Multiple	MD	Hetero	Multiple	BS	Hetero	Intra	Single	○
[136]	Two-Tier	Multiple	MD	Hetero	Multiple	BS, MD	Hetero	Intra	○	○
[28]	Three-Tier	Multiple	ND	ND	Multiple	General	Hetero	Intra	Single	○
[132]	Three-Tier	Multiple	MD	Hetero	Multiple	General, MD	Hetero	Intra	Multiple	●
[138]	Two-Tier	Single	MD	Homo	Single	BS	Homo	○	○	○
[42]	Two-Tier	Multiple	MD	Hetero	Multiple	BS, MS	Hetero	○	○	○
[45]	Two-Tier	Multiple	MD	Hetero	Multiple	General	Hetero	○	○	○
[149]	Two-Tier	Multiple	MD	Homo	Single	General	Homo	○	○	○

[53]	Two-Tier	Multiple	General	Hetero	Single	General	Homo	○	○	○
[64]	Three-Tier	Multiple	General	Hetero	Multiple	General	Hetero	Intra	Single	○
[128]	Three-Tier	Multiple	General	Hetero	Multiple	General	Hetero	Intra	Single	○
[33]	Three-Tier	Multiple	General	Hetero	Multiple	General	Homo	Intra	Single	○
[19]	Two-Tier	Multiple	MD	Hetero	Multiple	BS, Femto	Hetero	○	○	○
[11]	Two-Tier	Multiple	MD	Hetero	Multiple	BS	Hetero	○	○	○
[12]	Three-Tier	Multiple	General	ND	Multiple	General	Hetero	○	Single	○
[21]	Two-Tier	Multiple	General	Hetero	Multiple	General	Hetero	○	○	○
[74]	Two-Tier	Multiple	General	Hetero	Multiple	BS	Hetero	○	○	○
[30]	Three-Tier	Single	General	Hetero	Single	Rpi	Homo	Intra	Single	○
[155]	Two-Tier	Multiple	General	Hetero	Multiple	Hybrid	Homo	ND	○	○
[135]	Two-Tier	Multiple	General	Hetero	Single	General	Homo	○	○	○
[110]	Two-Tier	Multiple	General	Hetero	Single	General	Homo	○	○	○
[88]	Two-Tier	Multiple	General	Hetero	Multiple	AP	Hetero	○	Single	○
[142]	Three-Tier	Single	ND	ND	Single	General	Homo	○	Single	○
[24]	Three-Tier	Multiple	General	Hetero	Multiple	General	Hetero	Intra	Single	○
[49]	Three-Tier	Multiple	General	Hetero	Multiple	BS	Hetero	Intra	Single	○
[145]	Three-Tier	Multiple	General	Hetero	Single	Cloudlet	Homo	○	Single	○
[57]	Three-Tier	Multiple	MD	Hetero	Multiple	General	Homo	○	Single	○
[80]	Three-Tier	Multiple	General	Hetero	Multiple	General	Hetero	○	Single	○
[94]	Three-Tier	Multiple	General	Hetero	Multiple	General	Hetero	○	Single	○
[124]	Three-Tier	Single	General	Hetero	Multiple	General	Hetero	Intra	Single	○
[78]	Three-Tier	Multiple	General	Hetero	Multiple	General	Hetero	Intra	Single	○
[81]	Three-Tier	Multiple	General	Hetero	Multiple	General	Hetero	○	○	○
[27]	Three-Tier	Multiple	MD	Hetero	Multiple	AP	Hetero	○	Single	○
[55]	Two-Tier	Multiple	MD	Homo	Single	AP	Homo	○	○	○
[89]	Two-Tier	Single	General	Homo	Multiple	General	Hetero	○	○	○
[20]	Two-Tier	Single	MD	Hetero	Multiple	BS	Homo	○	○	○
[56]	Two-Tier	Multiple	MD	Hetero	Single	General	Homo	○	○	○
[73]	Two-Tier	Multiple	MD	Hetero	Single	General	Homo	○	○	○
[144]	Three-Tier	Multiple	MD	Hetero	Multiple	General	Hetero	○	Single	○
[31]	Many-Tier	Multiple	General	Hetero	Multiple	General	Hetero	○	Single	○
[72]	Three-Tier	Multiple	General	Hetero	Multiple	General	Hetero	Intra	Single	○
[38]	Many-Tier	Multiple	General	Hetero	Multiple	General	Hetero	Intra, Inter	Single	○
[141]	Two-Tier	Multiple	MD	Hetero	Multiple	BS	Hetero	○	○	○
[100]	Two-Tier	Multiple	MD	Hetero	Multiple	BS	Hetero	Intra	○	○
[119]	Three-Tier	Single	MD	Homo	Multiple	Hybrid	Hetero	○	Single	○
[133]	Three-Tier	Multiple	MD	Hetero	Multiple	AP	Hetero	○	Single	○
[71]	Two-Tier	Multiple	MD	Hetero	Multiple	BS	Hetero	Intra	○	○
[148]	Two-Tier	Multiple	Vehicle	ND	Multiple	AP	Hetero	Intra	○	○
[109]	Two-Tier	Multiple	Vehicle	Hetero	Multiple	BS	Hetero	Intra	○	○
[157]	Two-Tier	Multiple	Vehicle	Hetero	Multiple	BS, Vehicle	Hetero	Intra	○	○
[153]	Two-Tier	Single	MD	Homo	Multiple	General	Hetero	ND	○	○
[111]	Many-Tier	Multiple	MD	Hetero	Multiple	General	Hetero	ND	Single	○
[120]	Two-Tier	Multiple	General	Hetero	Multiple	General	Hetero	○	○	○
[143]	Two-Tier	Single	Vehicle	Hetero	Multiple	AP, Vehicle	Hetero	Intra	○	○
[83]	Many-Tier	Multiple	General	Hetero	Multiple	General	Hetero	ND	Single	○
[117]	Two-Tier	Multiple	General	Hetero	Multiple	General	Hetero	ND	ND	○
[90]	Three-Tier	Multiple	General	Hetero	Multiple	General	Hetero	ND	Multiple	○
[14]	Three-Tier	Multiple	General	Hetero	Multiple	General	Hetero	○	Single	○
[58]	Three-Tier	Multiple	General	Hetero	Multiple	General	Hetero	○	Single	○
[114]	Two-Tier	Multiple	Vehicle	Hetero	Multiple	General, Vehicle	Hetero	Intra	○	○
[6]	Three-Tier	Multiple	General	Hetero	Multiple	General	Hetero	○	Single	○
[16]	Three-Tier	Multiple	General	ND	Multiple	General	Hetero	Intra	Single	○
[22]	Two-Tier	Multiple	General	Homo	Multiple	General	Hetero	Intra	○	○
[4]	Three-Tier	Multiple	General	Hetero	Multiple	General	Hetero	○	Single	○
[50]	Three-Tier	Multiple	General	ND	Multiple	General	Hetero	○	Single	○
[59]	Three-Tier	Multiple	General	Hetero	Multiple	General	Hetero	○	Single	○
[61]	Two-Tier	Multiple	General	ND	Multiple	General	Hetero	Intra	○	○
[65]	Three-Tier	Multiple	General	Hetero	Multiple	General	Hetero	○	Single	○
[69]	Two-Tier	Multiple	General	Hetero	Multiple	General	Hetero	Intra	○	○
[82]	Three-Tier	Multiple	General	Hetero	Multiple	BS	Hetero	○	Single	○
[95]	Three-Tier	Multiple	General	Hetero	Multipe	General	Hetero	○	Single	○
[101]	Many-Tier	Multiple	General	Hetero	Multipe	General	Hetero	Intra	Single	○
[98]	Three-Tier	Multiple	General	ND	Multipe	General	Hetero	○	Single	○
[103]	Many-Tier	Multiple	General	Hetero	Multipe	General	Hetero	Intra	Single	○
[104]	Three-Tier	Multiple	General	Hetero	Multipe	General	Hetero	Intra	Multiple	●

[108]	Three-Tier	Multiple	General	ND	Multipe	General	Hetero	Intra	Single	○
[115]	Three-Tier	Multiple	ND	ND	Multipe	General	Hetero	ND	Single	○
[116]	Three-Tier	Multiple	General	Hetero	Multipe	General	Hetero	ND	Single	○
[134]	Three-Tier	Multiple	MD	Hetero	Single	General	Homo	○	Single	○
[46]	Three-Tier	Multiple	General	Hetero	Multiple	General	Hetero	Intra	Single	○
[102]	Three-Tier	Single	MD	Homo	Multiple	General	Hetero	Intra	Single	○

Hetero: Heterogeneity/Heterogeneous, Homo: Homogeneous, Coop: Cooperation, MD: Mobile Device, BS: Base Station, MS: Macrocell Station, AP: Access Point, RSU: Roadside Units, Femto: Femtocell, ● : Yes, ○ : No

5 OPTIMIZATION CHARACTERISTICS

Considering the application structure, environmental parameters, and the target objectives, each proposal formulates the problem of scheduling IoT applications in Fog computing. Optimization parameters directly affect the selection process and properties of suitable decision engines. Fig. 6 presents the principal elements in optimization characteristics, as described in what follows:

5.1 Main Perspective

The proposals in the literature can be divided into three categories based on their main optimization goal, namely IoT devices/users, system, and hybrid, which are described in the following:

5.1.1 IoT devices/users. The main perspective of several proposals is to satisfy the requirements of IoT applications such as minimizing their execution time and energy consumption of IoT devices, or improving user experience in terms of QoS and Quality of Experinece (QoE). Several works have considered IoT perspective for the optimization such as [13, 73, 114, 114, 124, 134].

5.1.2 System. The main perspective of this category is to improve the efficiency of resource providers such as minimizing their energy consumption, improving resource utilization, and maximizing the monetary profit [12, 19, 28, 50]. Hence, these works often assume IoT devices with very limited limited computational resources that transfer sensed data to the surrogate servers for processing and storage.

5.1.3 Hybrid. Some proposals targeted optimizing the parameters of both IoT devices/users and resource providers, referred to as hybrid optimization [55, 82, 98, 144]. In these works, IoT devices have some computational resources to serve their partial/complete tasks. However, they may send their requests to other surrogate servers if overall global parameters of IoT devices and systems can be optimized.

5.2 Objective Number

According to the number of main optimization objectives of each proposal, we classify the current literature into 1) **single objective** and 2) **multi objective** proposals. Multi objective proposals consider several parameters to simultaneously optimize them, incurring higher complexity. In the literature, several proposals targeted single objective optimization such as [56, 69, 114, 157], while other proposals try to optimize several parameters such as [38, 46, 95, 145].

5.3 Parameters

According to the main objectives and the nature of optimization parameters in the literature, we categorize these parameters into the following categories:

5.3.1 Time. One of the most important optimization parameters is the execution time of IoT applications. Minimizing the execution time of IoT applications provides users with a better QoS and QoE. This category contains any metrics related to time such as response time, execution time, and makespan used in the literature such as [24, 45, 103, 135].

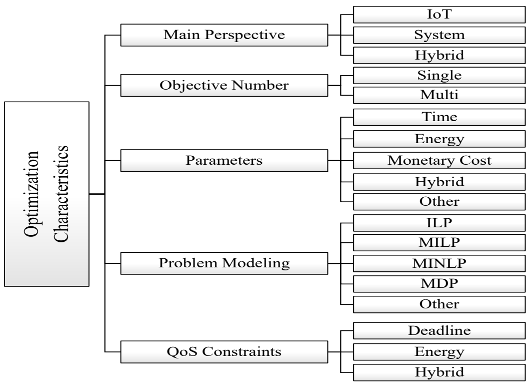

Fig. 6. Optimization characteristics taxonomy

5.3.2 Energy. IoT devices are usually considered as battery-limited devices. Hence, minimizing their energy consumption is one of the most important optimization parameters. Besides, energy consumption from FSs' perspective is two-fold. First, some FSs, similar to IoT devices, are battery-constrained, making optimizing the energy consumption of FSs an important challenge. Second, from the system perspective, the energy consumption of FSs should be minimized to reduce carbon emissions. This category contains any proposals considered energy consumption as an optimization parameter either from IoT devices or system perspectives such as [13, 54, 59, 110].

5.3.3 Monetary Cost. This category studies the proposals that have considered the monetary aspects either from IoT users (i.e., minimizing monetary cost) or system perspectives (i.e., increasing monetary profit) [12, 74, 94, 94, 108].

5.3.4 Other. Some works have considered other optimization parameters such as the number of served requests, system utility, and resource utilization, just to mention a few, such as [19, 28, 37, 115].

5.3.5 Hybrid. Several works also have considered a set of optimization parameters, referred as hybrid. These works use any combination of above-mentioned parameters simultaneously [39, 40, 46, 134].

5.4 Problem Modeling

Considering the main goal and optimization parameters, the optimization problem can be modeled/formulated. Considering surveyed literature in terms of the problem modeling approach, we classify the works into the following categories:

5.4.1 Integer Linear Programming (ILP). It is a problem type where the variables and constraints are all integer values, and the objective function and equations are linear. Several works have used ILP for problem modeling such as [74, 90, 97, 136].

5.4.2 Mixed Integer Linear Programming (MILP). In these problems, only some of the variables are constrained to be integers, while other variables are allowed to be non-integers. Also, the objective function and equations are linear. Several works have modelled their problem as an MILP such as [55, 70, 104, 145].

5.4.3 Mixed Integer Non-Linear Programming (MINLP). It refers to problems with integer and non-integer variables and non-linear functions in the objective function and/or the constraints. Several works such as [22, 84, 147, 156] have used MINLP to present their optimization problems.

5.4.4 Markov Decision Process (MDP). It provides a mathematical framework to model and analyzes problems with stochastic and dynamic systems. Several works have used MDP to model scheduling problem in Fog computing such as [9, 39, 115, 134].

5.4.5 Other. There are also some other optimization modeling approaches in the literature of scheduling applications in Fog computing such as game theory [49], lyapunov [19, 100], and mixed integer programming [6, 116].

5.5 QoS Constraints

The formulated optimization problem usually contains several constraints, incurring higher complexity compared to unconstrained problems. In this work, we classify techniques based on the QoS-related constraints applied to the main formulated problem into 1) **deadline** such as [32, 128, 140], 2) **energy** such as [11, 146], and 3) **hybrid** (i.e., any combination of deadline, energy, and cost) such as [22, 46, 100].

5.6 Discussion

In this section, we discuss the effects of identified optimization characteristics' elements on the decision engine and describe the lessons that we have learned. Besides, we identify several research gaps accordingly. Table 4 provides a summary of characteristics related to optimization problems in Fog computing.

5.6.1 ***Effects on the decision engine.*** The optimization characteristics affect the decision engine in various aspects, as briefly described below.

1. Objective number and parameters: Simultaneous optimization of multi-objective problems usually incur higher complexity for the decision engine. Also, when the number of key parameters in a multi-objective scheduling problem increases, finding the best parameters' coefficients becomes a critical yet challenging step.

2. Problem modeling: It can affect the choice of placement technique as some specific algorithms and techniques can be used to solve the scheduling problem. For example, several traditional LP and ILP tools and libraries exist to solve LP and ILP scheduling problems.

3. QoS constraints: They incur hard or soft constraints and limitations on the main objective/objectives of the scheduling problem, which intensify the complexity of the scheduling problem. The decision engine should satisfy these constraints either using classic Constraint Satisfaction Problem (CSP) techniques or using customized approaches.

5.6.2 ***Lessons learned.*** Our findings regarding the optimization characteristics in the surveyed works are briefly described in what follows:

1. The main perspective of optimization for almost 75% of works is IoT, while for the rest of the works is hybrid and system by 15% and 10%, respectively. The main perspective element affects how some metrics are evaluated. For example, when evaluating the energy consumption in the IoT perspective, the energy consumed by the surrogate servers for the execution of tasks is overlooked. However, in the system and hybrid perspectives, the energy consumption of all resource providers and all entities in the systems are evaluated, respectively.

2. Considering objective numbers in the optimization problem, the works are almost equally divided into single and multiple objective numbers. Overall, the majority of works studied time and/or energy as their main optimization metrics. While the works with an IoT perspective follow the same trend for the optimization metrics, the proposals with a system perspective almost consider the cost as their main optimization parameter. Also, the hybrid perspective proposals often consider a combination of time, energy, and/or cost as their main optimization metrics.

3. In problem modeling, the greater number of works have used either MDP or MINLP (each with roughly 25% of proposals) to formulate their problem. Also, some works initially had modeled their work as MINLP and then defined the MDP accordingly, such as [22, 56]. The rest of the works have used MILP (almost 15%), ILP (almost 15 %), and other optimization modeling approaches.

4. Almost 25% of works defined single or multiple QoS constraints for their problem, among which 90% have considered a single constraint, and the rest went for two QoS constraints. Among the QoS constraints, the deadline by 90% is the most used constraint in all works.

5.6.3 ***Research Gaps.*** We have identified several open issues for further investigation that are discussed below:

1. The main part of works in the literature either consider optimization problems from IoT devices/users. However, only a few works have considered IoT and system perspectives simultaneously (i.e., hybrid). Optimizing either of these perspectives can negatively affect other perspectives. To illustrate, when the principal target is minimizing the energy consumption of IoT devices, the majority of components or tasks are placed at FSs or CSs. However, it may negatively affect the energy consumption of resource providers and even increase the aggregated energy consumption in the environment. Hence, further investigation on hybrid optimization perspectives and mutual effects of different perspectives is required.

2. The cooperation among the resource providers (i.e., FSs, CSs) is an essential factor in offering higher-quality services. Proposals in the system and hybrid perspectives can also consider other metrics such as trust and privacy index for resource providers and study how they affect the overall performance.

3. QoS constraints are set to guarantee a minimum service level for end-users. In current literature, most of proposals have focused on the deadline as the constraint. However, several other parameters such as privacy, security, and monetary cost and their combination as hybrid QoS constraints are not studied.

6 DECISION ENGINE CHARACTERISTICS

The requirements of IoT applications in Fog computing can be satisfied if incoming IoT requests can be accurately scheduled based on the characteristics of application structure, environmental architecture, and optimization problems by the decision engine. The main responsibilities of the decision engine are organizing received IoT requests and solving the optimization problem through a placement decision while considering contextual information. Fig. 7 presents the main elements in decision engine, as described in what follows:

6.1 Deployment Layer

Decision engines can be deployed on servers at different layers unless the servers do not have sufficient resources to host them. Based on the deployment layer of the decision engine, the current literature can be classified into:

Table 4. Summary of existing works considering optimization Characteristics taxonomy

Ref	Optimization Characteristics					Ref	Optimization Characteristics				
	Main Persp	Object Number	Metrics	Prob Model	QoS Const		Main Persp	Object Number	Metrics	Prob Model	QoS Const
[9]	System	Single	Cost	MDP	○	[81]	IoT	Single	Time	ILP	Deadline
[13]	IoT	Single	Energy	MINLP	○	[27]	IoT	Multiple	Time, Cost	MINLP	Deadline
[18]	IoT	Multiple	Time, Energy, Cost	IP	○	[55]	Hybrid	Multiple	Energy, Time	MILP	○
[25]	IoT	Single	Cost	MINLP	Deadline	[89]	IoT	Multiple	Enery, Time	MDP	○
[39]	IoT	Multiple	Time, Energy, Weighted Cost	MDP	○	[20]	IoT	Multiple	Energy, Time	MDP	○
[40]	IoT	Multiple	Time, Energy, Weighted Cost	MILP	○	[56]	IoT	Single	Energy	MINLP, MDP	○
[84]	IoT	Single	Time	MINLP	○	[73]	IoT	Multiple	Energy, Time	MDP	○
[70]	IoT	Single	Time	MILP	○	[144]	Hybrid	Multiple	Time, Resource Utilization	MDP	○
[129]	Hybrid	Multiple	Time, Energy, Cost	MDP	○	[31]	Hybrid	Multiple	Time, Cost	MDP	Deadline
[156]	IoT	Single	Time	MINLP	○	[72]	Hybrid	Multiple	Energy, Cost	MDP	○
[47]	IoT	Single	Time	MDP	○	[38]	IoT	Multiple	Time, Eneegy	ILP	○
[54]	IoT	Multiple	Time, Comput Ratio	MDP	○	[141]	IoT	Single	Time	ND	○
[147]	IoT	Multiple	Time, Energy, Weighted Cost	MINLP	Deadline	[100]	Hybrid	Single	Time	Lyapu	Cost, Deadline
[137]	IoT	Single	Time	MDP	○	[119]	IoT	Single	Time	ND	○
[140]	Hybrid	Single	Energy	MINLP	Deadline	[133]	Hybrid	Multiple	Time, Energy	MINLP	Deadline
[97]	IoT	Single	Energy	ILP	○	[71]	IoT	Multiple	Time, Energy	MINLP	Deadline
[125]	IoT	Single	Time	MDP	○	[148]	IoT	Single	Time	MINLP	Deadline
[127]	IoT	Multiple	Time, Cost	ND	○	[109]	IoT	Single	Time	MDP	○
[32]	Hybrid	Single	Time/ Enegy	MILP	Deadline	[157]	IoT	Single	Time	MILP	Quality Loss
[146]	IoT	Multiple	Time, Energy	MINLP	Energy	[153]	IoT	Single	QoS	ND	○
[136]	IoT	Multiple	Cost, Time	ILP	○	[111]	IoT	Multiple	ND	ND	○
[28]	System	Single	Served Requests	MILP	○	[120]	IoT	Multiple	Time, Energy	ND	○
[132]	IoT	Single	Time	MINLP	○	[143]	IoT	Single	Time	MDP	○
[138]	IoT	Single	Time	MDP	○	[83]	IoT	Single	Time	ND	○
[42]	IoT	Single	Energy	MINLP	Deadline	[117]	IoT	Single	Bandwidth	ND	○
[45]	IoT	Single	Deadline	LP	○	[90]	IoT	Multiple	Time, Cost, Weighted Cost	ILP	○
[149]	IoT	Single	Time	MINLP	○	[14]	IoT	ND	Time	ND	○
[53]	IoT	Multiple	Time, Energy	MINLP, MDP	○	[58]	IoT	Single	Time	ND	○
[64]	Hybrid	Multiple	Time, Enery, Cost	ND	○	[114]	IoT	Single	Time	ND	○

[128]	IoT	Single	Offloaded Task	MILP	Deadline	[6]	IoT	Single	Time	MIP	○
[33]	Hybrid	Multiple	Time, Energy	MILP	○	[16]	IoT	Single	Time	ND	○
[19]	System	Single	Stystem Utility	Lyapu	○	[22]	IoT	Multiple	Time, Energy	MINLP, MDP	Deadline, Energy
[11]	Hybrid	Single	Time	IP	Energy	[4]	IoT	Single	Time	ND	○
[12]	System	Single	Cost	MILP	○	[50]	System	Multiple	Time, Energy	ILP	○
[21]	IoT	Single	Cost	MILP	Deadline	[59]	System	Single	Energy	ND	Deadline
[74]	System	Single	Cost	ILP	○	[61]	IoT	Single	Time	ND	○
[30]	IoT	Single	Time	ND	○	[65]	IoT	Single	Time	ND	○
[155]	IoT	Single	Time	ND	○	[69]	IoT	Single	Time	MINLP	○
[135]	IoT	Single	Time	MDP	○	[82]	Hybrid	Multiple	Time, Cost	MINLP	○
[110]	IoT	Single	Energy	MDP	Deadline	[95]	Hybrid	Multiple	Time, Energy, Cost	ND	Deadline
[88]	IoT	Single	Time	ND	Deadline	[101]	IoT	Single	Time	ND	○
[142]	IoT	Multiple	Time, Energy	ILP	○	[98]	hybrid	Multiple	Time, Energy, Cost	ND	Deadline
[24]	IoT	Single	Time	ND	○	[103]	IoT	Single	Time	ND	○
[49]	IoT	Multiple	Time, Energy	Game Theory	○	[104]	IoT	Multiple	Time, Energy	MILP	Deadline
[145]	IoT	Multiple	Time, Energy, Weighted Cost	MILP	○	[108]	IoT	Single	Cost	ILP	Deadline
[57]	IoT	Multiple	Time, Energy	MINLP	○	[115]	Hybrid	Multiple	Served Requests, Fog Numbers	MDP	○
[80]	IoT	Single	QoS	ILP	Cost, Deadline	[116]	IoT	Single	Time	MIP	○
[94]	IoT	Single	Cost	Lyapu	Deadline	[134]	IoT	Multiple	Time, Energy, Weighted Cost	MDP	○
[124]	IoT	Single	Time	ND	○	[46]	IoT	Multiple	Time, Energy, Weighted Cost	MDP	Deadline, Energy
[78]	System	Multiple	Time, Resource	ILP	○	[102]	IoT	Multiple	Cost, Energy	ND	Deadline

Main Pers: Main Perspective, Object number: Objective Number, Prob Model: Problem Model, QoS Const: QoS Constraints
Cost: Monetary Cost, MDP: Markov Decision Process, ILP: Integer Linear Programming, MINLP: Mixed Integer Non-Linear Programming, MIP: Mixed Integer Programming, MILP: Mixed Integer Linear Programming, ND: Not Defined, Lyapu: Lyapunov, ○: No

6.1.1 IoT Layer. The IoT devices usually are considered as resource-limited and battery-constrained devices. Hence, decision engines running on IoT devices should be very lightweight even with compromising the accuracy. In the literature, several works such as [56, 88, 97, 125] deployed decision engines at the IoT layer.

6.1.2 Fog/Edge Layer. Distributed FSs with sufficient resources situated in the proximity of IoT devices are the main deployment targets for the decision engines. They provide low-latency and high-bandwidth access to decision engines for IoT devices. Majority of works such as [39, 64, 89, 102] deployed the decision engines in Edge/Fog Layer.

6.1.3 Cloud Layer. CSs are potential targets for the deployment of decision engines. Although the access latency to CSs is higher, they provide high availability, making them a suitable deployment target where FSs are not available or when IoT applications are insensitive to higher startup time. Some works such as [27, 119] considered cloud layer for the deployment of decision engines.

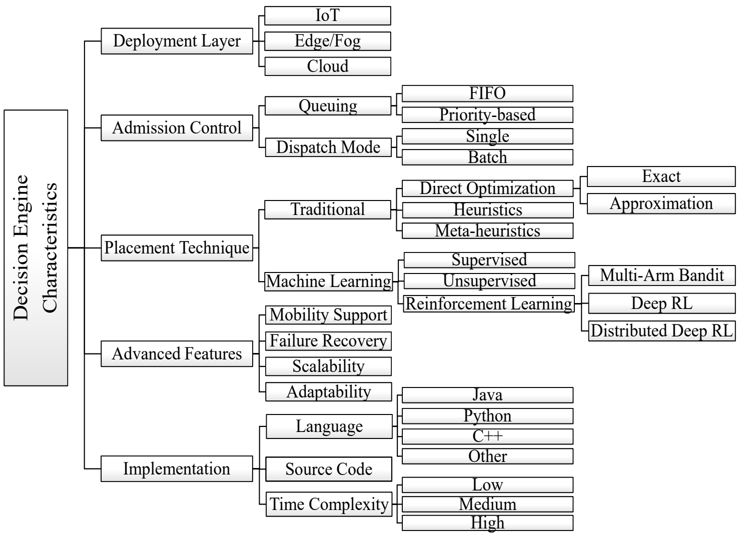

Fig. 7. Decision engine taxonomy

6.2 Admission Control

The admission control presents the behavior of decision engines when new requests arrive. It denotes how the new requests are queued and organized by the dispatching module for placement.

6.2.1 Queuing. Decision engine may use different queuing policies when incoming IoT requests arrives. Based on queuing policy, we classify works into 1) **First-in-First-Out (FIFO)** such as [32, 39, 125] and 2) **Priority-based** where incoming requests are sorted based on their priority (e.g., deadline) [45, 50, 128].

6.2.2 Dispatching Mode. The dispatching module forwards requests from input queue to the placement module. Based on the selection policy of dispatching module, current literature can be classified to 1) **single** model where only one task at a time is dispatched for placement [22, 47, 143] and 2) **batch** model where a set of tasks are forwarded to placement module [4, 40, 98].

6.3 Placement Technique

Placement technique is the actual algorithm used to solve the optimization problem. Each placement algorithm has its advantages and disadvantages. Hence, it should be carefully selected based on the properties and dynamics of applications, users, environment, and deployment layer. We classify placement techniques based on their approach to find the solution into two broad categories:

6.3.1 Traditional. In this approach, the programmer/designer defines the required logic of policies for the placement technique. The traditional placement technique can be further divided into three subcategories:

1. **Direct Optimization**: In this category, the optimization problem will be solved using classical optimization tools either using 1) **Exact** approach to find the optimal solution such as [18, 25] or 2) **Approximation** approach to find a near optimal solution such as [11, 84, 136].

2. **Heuristics**: These algorithms are a set of typically problem-dependent algorithmic steps to find a feasible solution for the problem. Heuristics usually scale well as their TC is low while they do not guarantee finding the optimal solution of the problem [13, 14, 28, 70].

3. **Meta-heuristics**: Meta-heuristics are composed of several advanced heuristics and typically are problem-independent, such as Genetic Algorithm (GA) and Simulated Annealing (SA). Although these algorithms usually perform better than heuristics, similarly they cannot guarantee to find the optimal solution. Several works such as [33, 40, 90, 132] used meta-heuristics.

6.3.2 Machine Learning (ML). ML is a family of algorithms that can learn the required policies for placement techniques from historic data. ML algorithms scale reasonably well, however, they require accurate and ideally large samples of historic data. The ML-based placement techniques can be further divided into three subcategories:

1. **Supervised Learning**: These algorithms learn by using labeled data as its input. Type of problems are regression and classification, and some of the algorithms are linear regression and logistic regression. Some works in the current literature used supervised ML for the placement technique such as [57, 146, 147].

2. **Unsupervised Learning**: These algorithms are trained using unlabelled data, contrary to supervised ML, without any guidance. Some unsupervised algorithms are K–Means and fuzzy C–Means. Some works in the current literature used unsupervised ML for the placement technique such as [117, 120, 143].

3. **Reinforcement Learning (RL)**: In these algorithms, agent/agents learn the required policy for placement technique by interaction with an uncertain and potentially complex environment. It does not require pre-defined data, and type of problems are exploitation or exploration. The current RL-based literature in scheduling IoT applications can be divided into 1) **Multi-Armed Bandit (MAB)** which are among the simplest RL problems such as [127, 156], 2) **Deep RL (DRL)** where deep learning is used in RL such as [9, 89, 125], and 3) **Distributed DRL** where several agents works collaboratively in a distributed manner for efficient learning such as [22, 39, 129].

6.4 Advanced Features

To fully utilize the potential of the Fog computing paradigm, several advanced features can be augmented with decision engines to capture high dynamics of this paradigm, described below:

6.4.1 Mobility Support. A significant number of IoT devices are moving entities (e.g., vehicles), requiring connected service through their path. So, decision engines should manage the migration process of application components and find suitable surrogate servers accordingly. Several works such as [11, 38, 90, 138] address mobility and migration management challenges alongside scheduling IoT applications.

6.4.2 Failure Recovery. In highly dynamic systems such as Fog computing, failure may happen due to software or hardware-related issues. So, application components faced with failure should be re-executed. Some works consider failure recovery mechanisms in their decision engines such as [16, 38, 40].

6.4.3 High Scalability. As a large number of IoT devices and servers exist in the Fog environment, mechanisms and algorithms used in the decision engine should be highly scalable and provide well-suited performance when the system size grows. Several works have studied the scalability feature of their techniques when the number of IoT applications and servers increases or discussed how their distributed techniques work efficiently in large systems such as [30, 53, 73].

6.4.4 High Adaptability. This feature ensures that the decision engine dynamically captures the contextual information (i.e., application, environment, etc), and updates the policies of placement techniques accordingly. In Fog literature, several works such as [31, 39, 108, 153] offer solutions with high adaptability.

6.5 Implementation

The implementation characteristics of decision engines are studied based on the following criteria:

6.5.1 Language. Different programming languages are used for the implementation of decision engines, while the majority have used Python [39, 74], Java [40, 119], and C++ [12, 124].

6.5.2 Source Code. Open-source decision engines help researchers to understand the detailed implementation specifications of each work, and minimize the reproducibility effort of decision engines, especially for comparison purposes. Some works such as [24, 110, 134] have provided the source code repository of their decision engines.

6.5.3 Time Complexity (TC). TC of each placement technique presents the required time to solve the optimization problem in the worst-case scenario. It directly affect the service startup time and the decision overhead of each technique. Based on the current literature, we classify the TC into 1) **Low** the solution of optimization problem can be obtained in polynomial time where the maximum power of variable is equal or less than two (i.e., $O(n^2)$) [9, 16, 39], 2) **medium** where the time complexity is polynomial and the maximum power is less than or equal to 3 (i.e., $O(n^3)$) [13, 70, 104], and 3) **High** for exponential TC and polynomials with high maximum power [18, 25, 128].

6.6 Discussion

In this section, we describe the lessons that we have learned regarding identified elements in decision engine characteristics of the current literature. Besides, we identify several research gaps accordingly. Table 5 provides a summary of decision engines-related characteristics in Fog computing.

6.6.1 ***Lessons learned.*** Our findings regarding the decision engine characteristics in the surveyed works are briefly described in what follows:

1. Almost 85% of surveyed works deployed the decision engine at the Edge layer in the proximity of IoT devices. Since the Edge servers can be accessed with lower latency and higher access bandwidth, deployment of decision engines at the Edge can reduce the startup of IoT applications. However, the Edge devices should have sufficient resources to run the decision engine. Some proposals (about 10%) also deployed the decision engine on IoT devices. Deployment of a decision engine on IoT devices provides more control for IoT devices, especially mobile ones. It eliminates the extra overhead of communication with surrogate servers for making a decision. However, IoT devices often have very limited resources that are incapable of running powerful decision engines.

2. The queuing is an important element in the admission control that almost 80% of the works have not studied. Since most of works have considered several IoT devices in the environment, several IoT requests may arrive in each decision time-slot with a high probability. Hence, different queuing models can dramatically affect the decision engine performance and the QoS of end-users. FIFO and priority queue share the same proportion of proposals among the works that mentioned their queuing policy. Also, in priority-based queuing, almost all works have considered the deadline of applications or tasks as their main priority metric. Moreover, for the policy of dispatching module, about 75% of works selected single dispatching while 25% of works studied batch dispatching policy. Since different IoT requests may arrive in the same decision time-slot, the batch dispatching policy helps study the mutual effects of IoT applications with diverse resource requirements in the placement decision.

3. The traditional placement techniques are used in almost 60% of the proposals, while the ML-based placement techniques are studied in the rest of the works (almost 40%). However, the number of ML-based placement techniques has significantly increased in recent years. In traditional placement techniques, direct optimization, heuristics, and meta-heuristics share the same proportion of proposals. Also, in meta-heuristics techniques, the majority of works used population-based meta-heuristics, especially different variations of the GA. In the ML-based techniques, the majority of proposals have used RL-based techniques (almost 70%), specially DRL. Moreover, in the DRL techniques, the larger number of works used centralized DRL techniques such as DQN. However, the exploration and convergence rate of centralized DRL techniques are very slow. Thus, several studies have recently been conducted to adapt distributed DRL (i.e., DDRL) techniques for resource management in Edge/Fog computing environments, such as [39, 53, 73], to improve the exploration cost and convergence rate of the DRL techniques.

4. In advanced features, almost 25% of proposals embedded different mechanisms (i.e., traditional or ML-based) for the mobility management of IoT devices and migration of applications' constituent parts. Also, about 25% of studied works, mostly ML-based techniques, offer high adaptability features in their decision engine. However, traditional works often neglect to provide different mechanisms to support high adaptability. This is mainly because the scheduling policies are not statically defined in ML-based techniques. Hence, as the environmental or application properties change, the policies can be learned and updated accordingly. However, in the traditional scheduling techniques, updating the scheduling policies according to dynamic changes in environmental or application properties is very costly and time-consuming. Almost 20% of proposals studied different mechanisms to support high scalability feature, either using ML-based techniques or traditional approaches. In advanced features, the failure recovery mechanisms and techniques in scheduling are not well-studied and only a few works embedded these mechanisms in their scheduling techniques.

5. Considering the implementation of the techniques, almost 50% of the works mentioned their employed programming language. Java and python programming language are the most-employed programming language and are almost equally used in different proposals. However, Python is mainly used for ML-based techniques and direct optimization techniques, while Java is mostly used to implement traditional decision engines. Moreover, only about 10% of proposals shared their open-source repositories with researchers and developers among the surveyed works. Finally, about 65% of proposals discussed the TC of their works, among which almost 80% proposed decision engines with low TC while some proposals (almost 10%) went for medium TC and few works (almost 10%) proposed decision engines with high TC. The high TC proposals are among the direct optimization category of traditional approaches. While these high TC proposals cannot be currently adapted to large-scale Edge and Fog computing environments, they can find the optimal solution in small-scale problems. Hence, they can be used as a reference for the evaluation of other proposals.

6.6.2 *Research Gaps.*

We have identified several open issues for further investigation that are discussed below:

1. The admission control concept in terms of different queuing, dispatching, and their mutual effect is not well studied in the current literature. Also, the greater number of works consider a single task dispatching model and overlook batch placement of applications, especially for applications with dependent tasks.

2. While traditional placement techniques (e.g., heuristics, meta-heuristics) are studied well in the literature, ML-based techniques are still in their infancy. Due to the lack of a large number of datasets, supervised and unsupervised ML have not been thoroughly considered. Also, the majority of employed RL techniques are centralized approaches, neglecting collaborative learning of multiple distributed agents for better efficiency and lower exploration costs.

3. Although all servers and devices are prone to failures, among advanced features, failure recovery mechanisms, algorithms, and their integration with the placement technique is the least-studied concept. Even the best placement techniques cannot complete their process in real-world scenarios unless a suitable failure recovery mechanism is embedded.

4. In the surveyed works, there is no proposal to study all the four identified elements in the advanced features (i.e., mobility, failure recovery, scalability, and adaptability) and describe the behavior and mutual effects of these elements on each other and decision engine.

5. Among the studied literature, none of the works has studied the privacy problem from different perspectives, such as end-users' data privacy, resource providers' privacy, and the decision engine's mechanisms for improving privacy.

7 PERFORMANCE EVALUATION

Different approaches and metrics have been used by the research community to evaluate the performance of their techniques. Identifying and studying these parameters helps to select the best approach and metrics for the implementation of new proposals and fair comparisons with other techniques in the literature. Fig. 8 presents a taxonomy and the main elements of performance evaluation, described below:

7.1 Approaches

The performance evaluation approaches can be divided into four categories, namely analytical, simulation, practical, and hybrid. There are different important aspects to consider when selecting an approach for the evaluation of proposals, such as credibility, implementation time, monetary cost, reproducibility time, and scalability. Fig. 9 presents a qualitative comparison of different approaches used in performance evaluation.

7.1.1 Analytical. One of the popular approaches for the evaluation of different proposals is analytical tools. Usually, the implementation time, reproducibility time, and monetary cost of analytical tools are low, and scalable experiments can be executed. However, the credibility of such experiments is low since the dynamics of resources, applications, and environment cannot be fully captured and tested. Matlab is among the most popular tools that is either used directly [28, 147, 147] or integrated with some other libraries such as Sedumi[1] [116]. Also, C++ based analytical tools have been used in the literature, such as [12, 84].

[1]https://github.com/sqlp/sedumi

Table 5. Summary of existing works considering decision engine taxonomy

Ref	Decision Engine Characteristics										
	Deployment Layer	Admission Control		Placement Technique	Advanced Features				Implementation		
		Queuing	Dispatch		Mobility Support	Failure Recovery	High Scalability	High Adaptability	Language	Source Code	Time Complexity
[9]	Edge	ND	Single	ML, RL, DRL, (DQN)	○	○	○	●	ND	○	Low (MP 2)
[13]	Edge	ND	ND	Tr, H, Greedy	●	○	○	○	ND	○	Medium (MP 3)
[18]	Edge	ND	ND	Tr, DO, Exact	●	○	○	○	ND	○	High (Exp)
[25]	Edge	FIFO	Single	Tr, DO, Exact, (BB)	○	○	○	○	Java	○	High (Exp)
[39]	Edge	FIFO	Single	ML, RL, DDRL, (IMPALA)	○	○	●	●	Python	○	Low (MP 2)
[40]	Edge	FIFO	Batch	Tr, MetaH, (MA)	○	●	●	○	Java	○	Low (MP 2)
[84]	Edge	ND	Batch	Tr, DO, Approx	●	○	○	○	C++	○	ND
[70]	Edge	FIFO	Single	Tr, H	○	○	○	○	Java	○	Medium (MP 3)
[129]	Edge	ND	ND	ML, RL, DDRL, (A3C)	●	○	●	●	Java	○	Low
[156]	Edge	ND	Batch	ML, RL, MAB	○	○	○	●	ND	○	Low
[47]	Edge	ND	Single	ML, RL, DRL, (DQN)	○	○	○	●	ND	○	Low
[54]	Edge	ND	ND	ML, RL, DRL	○	○	○	●	ND	●	Low
[147]	Edge	ND	Single	ML, Sup, (DeepL)	○	○	○	○	ND	○	Low
[137]	Edge	ND	Single	ML, RL, (Q-learning)	●	○	○	●	ND	○	Low
[140]	Edge	ND	Single	ML, Sup, (Imitation)	●	○	○	○	ND	○	Medium
[97]	IoT	ND	Single	ND	○	○	○	○	Android/Java	●	ND
[125]	IoT	FIFO	Single	ML, RL, DRL (Double DQN)	○	○	●	●	ND	○	Low
[127]	IoT	ND	Single	ML, RL, MAB	●	○	●	●	ND	○	Low
[32]	Edge	FIFO	Single	Tr, H	○	○	○	○	Android/Java	○	Low
[146]	Edge	ND	Single	ML, Sup, (Imitation)	○	○	○	○	ND	○	Low
[136]	Edge	ND	Single	Tr, DO, Approx	●	○	○	○	ND	○	Low (MP 2)
[28]	Edge	ND	Single	Tr, H, Greedy	○	○	○	○	ND	○	High
[132]	Edge	ND	Batch	Tr, MetaH, (SA)	○	○	○	○	ND	○	ND
[138]	Edge	ND	ND	Tr, DO, Approx	●	○	○	○	ND	○	Medium
[42]	Edge	ND	Single	Tr, MetaH (GA-PSO)	○	○	○	○	ND	○	Low (MP 2)
[45]	Edge	Priority	Single	Tr, DO, Approx	○	○	●	○	ND	○	ND
[149]	Edge	ND	Single	Tr, H	○	○	○	●	ND	○	ND
[53]	Edge	ND	Single	ML, RL, DRL, DDRL	○	○	●	●	ND	○	Low
[64]	Edge	ND	Single	Tr, MetaH, (NSGA2)	●	○	○	○	Java	○	ND
[128]	Edge	Priority	Single	Tr, H	○	○	○	○	ND	○	High (MP 5)
[33]	Edge	ND	Batch	Tr, MetaH, (Ant Mating)	○	○	○	○	ND	○	ND
[19]	Edge	ND	ND	Tr, H	○	○	○	○	ND	○	ND
[11]	Edge	ND	Single	Tr, DO, Approx, (SAA)	●	○	○	○	ND	○	ND
[12]	ND	ND	Batch	Tr, H	○	○	○	○	C++	○	Low
[21]	Edge	ND	Single	Tr, DO, Approx	○	○	○	○	ND	○	Low (MP 2)
[74]	Edge	ND	Batch	Tr, DO, Approx	○	○	○	○	Python	○	Low (MP 2)
[30]	Edge	ND	Single	Tr, DO, Approx	○	○	○	●	Python	○	ND
[155]	Edge	ND	Single	Tr, DO, Approx	○	○	○	○	ND	○	ND
[135]	Edge	ND	Single	ML, RL, DRL, (PPO)	○	○	○	●	Python	○	Low

[110]	Hybrid	ND	Single	ML, RL, DDRL	○	○	●	●	Python	●	Low
[88]	IoT	ND	Single	Tr, DO, Approx	○	○	○	○	ND	○	ND
[142]	Edge	ND	Single	Tr, Other, (Min-cut)	○	○	○	○	Java	●	Low (MP 2)
[24]	Edge	FIFO	Single	Tr, MetaH, (GA)	○	○	●	●	Python	●	Low
[49]	Edge	ND	Single	Tr, DO, Approx	○	○	○	○	ND	ND	ND
[145]	Edge	ND	Batch	Tr, MetaH, (NSGA3)	○	○	○	○	Java	○	ND
[57]	IoT	ND	Single	ML, Sup, DDeepL	○	○	●	○	Python	●	Low
[80]	Edge	ND	Single	Tr, DO, Exact	○	○	○	○	Java	○	high
[94]	Edge	ND	Single	Tr, DO, Approx	○	○	○	○	Python	○	ND
[124]	Edge	ND	Single	Tr, H	○	○	○	○	C++	○	Low
[78]	Edge	ND	Single	Tr, H	○	○	○	○	Java	○	ND
[81]	Edge	ND	Single	Tr, H	○	○	○	○	Java	○	Low
[27]	Cloud	ND	Single	Tr, DO, Approx	○	○	○	○	ND	○	ND
[55]	IoT	ND	Single	ML, Sup, (DDeepL)	○	○	●	○	Python	○	Low
[89]	Edge	ND	Single	ML, RL, DRL, (DQN)	○	○	○	●	ND	○	Low
[20]	Edge	FIFO	Single	ML, RL, DRL, (DoubleDQN)	○	○	○	●	ND	○	Low
[56]	IoT	ND	Batch	ML, RL, DRL, (DQN)	○	○	○	●	ND	○	Low
[73]	Edge	ND	Single	ML, RL, DDRL, (D3PG)	○	○	●	●	ND	○	Low
[144]	Edge	FIFO	Single	ML, RL, DRL, (DQN)	○	○	○	●	Python	○	Low
[31]	Edge	ND	Single	ML, RL, DRL, (DoubleDQN)	○	○	○	●	Python	○	Low
[72]	ND	ND	Single	ML, RL, DRL, (DQN)	○	○	○	●	Java	○	Low
[38]	Edge	FIFO	Single	Tr, H	●	●	○	○	Java	●	Low (MP 2)
[141]	Edge	ND	Single	Tr, H	●	○	○	○	ND	○	ND
[100]	Edge	ND	Single	Tr, DO, Approx	●	○	○	○	Java	○	ND
[119]	Cloud	ND	Single	ML, Sup, (Gradient Tree Boosting)	●	○	○	○	Java	●	Low
[133]	Edge	ND	Single	Tr, MetaH, (GA)	●	○	○	○	ND	○	Low (MP 2)
[71]	ND	ND	Batch	Tr, MetaH, (GA)	●	○	○	○	ND	○	ND
[148]	IoT	FIFO	Batch	Tr, DO, Approx	●	○	●	○	ND	○	Low
[109]	IoT	ND	Batch	ML, RL, DDRL, (A3C)	●	○	●	●	ND	○	ND
[157]	Edge	ND	Batch	Tr, DO, Approx	●	○	○	○	ND	○	ND
[153]	Edge	ND	Single	ML, RL, DRL, (DQN)	●	○	○	●	Python	○	Low
[111]	Edge	ND	Single	ML, Sup, (Regression Tree)	○	○	○	○	Java	○	ND
[120]	ND	ND	Single	ML, Unsup, (AHP)	○	○	○	○	ND	○	ND
[143]	Edge	ND	Single	ML, Unsup, (Baum-Welch Algorithm),	●	○	○	○	ND	○	ND
[83]	Edge	ND	Batch	ML, Unsup, (K-means)	○	○	○	○	ND	○	ND
[117]	ND	ND	Single	ML, Unsup, (K-means)	○	○	○	○	ND	○	ND
[90]	Edge	ND	Single	Tr, MetaH, (Tabu)	●	○	○	○	ND	○	ND
[14]	Edge	Priority	Single	Tr, H, (Spring Algorithm)	○	○	○	○	ND	○	ND
[58]	Edge	ND	Batch	Tr, MetaH, (ACO)	○	○	○	○	ND	○	ND
[114]	Edge	ND	Batch	Tr, MetaH, (MA)	●	○	○	○	ND	○	Low (MP 2)
[6]	Edge	Priority	Batch	Tr, MetaH, (GA)	○	○	○	●	ND	○	ND
[16]	Edge	Priority	Single	Tr, H	○	●	○	○	Python	○	Low (MP 2)
[22]	IoT	ND	Single	ML, RL, DDRL, (DDPG)	○	○	●	●	Python	○	Low
[4]	Edge	ND	Batch	Tr, MetaH, (AEO)	○	○	○	○	ND	○	ND
[50]	Edge	Priority	Batch	Tr, MetaH, (GA)	○	○	○	○	ND	○	ND

[59]	Edge	ND	Single	Tr, H	○	○	○	○	ND	○	Low (MP 2)
[61]	Edge	Priority	Batch	Tr, MetaH, (PSO)	○	○	○	○	ND	○	Low
[65]	Edge	ND	Single	Tr, MetaH, (ACO)	○	○	○	○	ND	○	Low
[69]	Edge	ND	Batch	Tr, MetaH	○	○	○	○	Python	○	Low (MP 2)
[82]	Edge	ND	Batch	Tr, MetaH, (GA)	○	○	○	○	Python	○	Low
[95]	Edge	ND	Batch	Tr, MetaH, (GA)	○	○	○	○	Python	○	Low
[101]	Edge	FIFO	Single	Tr, H	○	○	○	○	Java	○	Medium (MP of 3)
[98]	Edge	ND	Batch	Tr, MetaH, (SSA)	○	○	○	○	Python	●	ND
[103]	Edge	Priority	Single	Tr, H	○	○	○	○	Java	○	ND
[104]	IoT	ND	Single	Tr, MetaH, (GA)	○	○	●	○	ND	○	Medium (MP 3)
[108]	Edge	Priority	Single	Tr, H, Greedy	○	○	●	○	Go	●	Low
[115]	Edge	Priority	Single	ML, RL, DRL, (DQN)	○	○	○	●	Python	○	Low
[116]	IoT	ND	Batch	Tr, DO, Approx	○	○	●	○	ND	○	Low
[134]	IoT	ND	Single	ML, RL, DRL	○	○	●	●	Python	●	Low
[46]	Edge	ND	Single	ML, RL, DRL, (DQN)	○	○	○	●	Python	○	ND
[102]	Edge	ND	Single	Tr, MetaH, (SPEA)	○	○	○	○	ND	○	Low (MP 2)

ND: Not Defined, ML: Machine Learning, Tr: Traditional, RL, Reinforcement Learning, DRL: Deep Reinforcement Learning, DDRL: Distributed Deep Reinforcement Learning,
MP: Max Power, H: Heuristics, MetaH: Metaheuristics, DO: Direct Optimization, BB: Branch and Bound, MA: Memetic Algorithm, FIFO: First-In-First-Out, Approx: Approximation,
MAB: Multi-Arm Bandit, A3C: Asynchronous Actor-Critic Agents, DeepL: Deep Learning, DDeepL: Distributed Deep Learning, Imitation: Imitation Learning, SA: Simulated Annealing,
GA: Genetic Algorithm, SAA: Sample Average Approximation, PPO: Proximal Policy Optimization, D3PG: Double-Dueling-Deterministic Policy Gradients, AHP: Analytic Hierarchy Process,
Sup: Supervised, Unsup: Unsupervised, ACO: Ant Colony Optimization, AEO: Artificial Ecosystem-based Optimization, Tabu: Tabu Search, PSO: Particle Swarm Optimization,
SSA: Sparrow Search Algorithm, SPEA: Strength Pareto Evolutionary Algorithms
●: Yes, ○: No

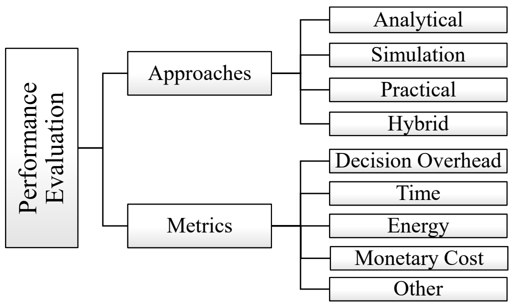

Fig. 8. Performance evaluation taxonomy

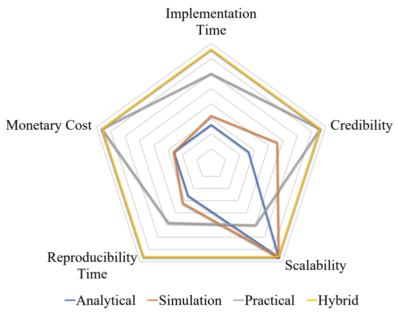

Fig. 9. Performance evaluation approaches

7.1.2 Simulation. Simulators keep the advantages of analytical tools while improving the credibility of evaluations by simulating the dynamics of resources, applications, and environments. In the literature, iFogSim [43, 77] is among the most popular simulators for Fog computing [40, 64, 72, 129]. Besides, several researchers have used Cloudsim [17] such as [25, 145] or SimPy[2] such as [31, 94] to simulate their scenarios in Fog computing.

7.1.3 Practical. The most credible approach for the evaluation of proposals is practical implementation. However, due to high monetary cost, implementation time, and reproducibility time, it is not the most efficient approach for different scenarios, especially evaluations requiring high scalability. In the literature, few works such as [11, 32, 117, 119] evaluated their proposals using small-scale practical implementations.

7.1.4 Hybrid. In this approach, researchers evaluate their proposals using practical implementations in small-scale and simulators or analytical tools in large-scale. Although implementation and reproducibility time of this approach is high, it provides high scalability and credibility. In the literature, few works such as [39, 74, 110] follow the hybrid approach.

7.2 Metrics

The metrics used in performance evaluation in Fog computing are directly or indirectly related to the optimization parameters and system properties. Based on the nature and popularity of these metrics in the literature, we categorize them into 1) **time** (e.g., deadline, response time, execution time, makespan) [24, 25, 70, 89], 2) **energy** (e.g., battery percentage, saved energy) [13, 20, 55, 55], 3) **monetary cost** (e.g., service cost, switching cost) [9, 90, 94, 100], and 4) **other** metrics (e.g., number of interrupted tasks, resource utilization, throughput, deadline miss ratio) [14, 16, 38, 49]. Also, we consider 5) **decision overhead** as an important evaluation metric to study the overhead of proposals (often in terms of time and energy), used in some works such as [39, 64, 102, 142].

7.3 Discussion

In this section, we describe the lessons that we have learned regarding identified elements in the performance evaluation of the current literature. Besides, we identify several research gaps accordingly. Table 6 provides a summary of characteristics related to performance evaluation in Fog computing.

7.3.1 ***Lessons learned.*** Our findings regarding the performance evaluation in the surveyed works are briefly described in what follows:

[2]https://simpy.readthedocs.io/en/latest/

1. More than half of the works used the simulation as their performance evaluation approach while 30% of the proposals used an analytical approach. The practical and hybrid approaches equally share the rest of 20% of the works. For the analytical approach, the most of works used Matlab or Python programming languages, while Java and Python are mostly used for the simulation approach. In practical and hybrid approaches, Java and Python are equally employed in proposals.

2. As the performance evaluation metric, time and its variations (e.g., response time, makespan) are used in more than 80% of the works. The second-highest-used metric is energy at 35%. However, the decision overhead and cost are only studied in 15% of the works. Besides, less than 5% of the proposals studied the performance of their scheduling technique using all the identified metrics.

7.3.2 Research Gaps. We have identified several open issues for further investigation that are discussed below:

1. Although the monetary costs of sensors and edge devices (e.g., Rpi, Jetson Platform) have reduced and they are highly available in different configurations, compared to a few years ago, the majority of proposals still consider analytical tools and simulators as their only approach for performance evaluation. While some works have considered practical and hybrid approaches for the performance evaluation of their work, further efforts are required to study the dynamics of the system, resource contention, and collaborative execution of the application in real environments, especially considering new machine learning techniques such as DRL and DDRL [39, 110].

2. The decision overhead of proposals has direct effects on users and resources in terms of the startup time of requested services and resource utilization. To illustrate, not only do healthcare applications require low response time, but they also need low startup time, especially for critical applications such as emergency-related applications (e.g., heart-attack prediction and detection). Also, the overhead of proposals can severely affect the resource usage and energy consumption of servers, especially battery-constrained ones. Among the techniques considered decision overhead as a metric, they mostly focus on time while other metrics (e.g., energy, cost) need further investigation.

8 SCHEDULING TECHNIQUE: IMPORTANT DESIGN OPTIONS

In this section, we discuss the real-world characteristics of application structure and environmental architecture and accordingly present several guidelines for designing a scheduling technique.

1. The number of IoT applications is constantly increasing in different application domains. The majority of these applications are defined as a set of dependent modules/services [155]. Besides, sharing and reusing modules/services for faster development and better management of applications is of paramount importance. Moreover, dependent IoT applications are usually modeled as a graph of tasks and their respective invocations. In this case, IoT applications with monolithic and independent design can also be defined as an application graph with only one module and an application graph with several modules where the size of invocations is zero, respectively. Hence, we consider IoT applications with dependent modules/services (i.e., modular and loosely-coupled categories) as the main architectural design choices in the application structure. Accordingly, the decision engine requires a component for identifying and satisfying the constraint among modules/services.

Table 6. Summary of existing works considering performance evaluation taxonomy

Ref	Performance Evaluation						Ref	Performance Evaluation					
		Metrics							Metrics				
	Approach	DO	T	E	C	O		Approach	DO	T	E	C	O
[9]	Sim (OPNET)	○	○	○	●	○	[81]	Hybrid (Sim, Prac) (iFogSim)	○	●	○	○	Resource Overhead
[13]	ND	○	○	●	○	○	[27]	Analytic	○	○	○	○	Weighted Cost
[18]	Analytic	○	●	●	●	○	[55]	Sim	●	●	●	○	○
[25]	Sim (Cloudsim)	○	●	○	●	○	[89]	Sim	○	●	●	○	Task Drop Rate
[39]	Hybrid (Sim, Prac), (iFogSim)	●	●	●	○	Weighted Cost	[20]	Sim	○	●	●	○	Task Drop Rate
[40]	Sim (iFogSim)	●	●	●	○	Weighted Cost	[56]	Sim	○	○	●	○	○
[84]	Analytic	○	●	○	○	Resource Utilization	[73]	Sim	○	●	●	○	Success Rate
[70]	Sim (Cloudsim)	○	●	○	○	○	[144]	Sim	○	●	○	○	Resource Utilization
[129]	Sim (iFogSim)	●	●	●	●	○	[31]	Sim (SimPy)	○	●	●	●	○
[156]	Sim	○	●	○	○	○	[72]	Sim (iFogSim)	○	●	●	●	Network Usage, Weighted Score
[47]	Sim	○	●	○	○	○	[38]	Sim (iFogSim)	●	●	●	○	Weighted Cost, Interrupted Tasks, Migrated Tasks
[54]	Sim	○	●	○	○	Computation ratio	[141]	Sim	○	●	○	○	○
[147]	Analytic (Matlab)	○	○	○	○	Weighted Cost	[100]	Sim (One Simulator)	●	●	○	●	○
[137]	Analytic (Matlab)	○	●	○	○	Migration cost	[119]	Prac	○	●	○	○	○
[140]	Analytic	○	●	●	○	○	[133]	Sim	○	○	○	○	Weighted Cost
[97]	Hybrid (Sim+Prac)	●	●	●	○	○	[71]	Analytic	○	○	○	○	Weighted Cost, Offloaded Tasks
[125]	Sim	○	●	○	○	Dropped Tasks	[148]	Analytic	○	●	○	○	○
[127]	Sim	●	●	○	○	Switching	[109]	Sim	○	●	○	○	○
[32]	Prac	○	●	●	○	Throughput	[157]	Sim	○	○	○	○	○
[146]	Sim	○	●	○	○	○	[153]	Sim	○	○	○	○	QoS
[136]	Analytic	○	○	○	○	Weighted Cost	[111]	Sim (Cloudsim)	○	●	●	○	○
[28]	Analytic (Matlab)	○	○	○	○	Satisfied Requests	[120]	Analytic (Matlab)	○	●	●	○	Offloaded Tasks, Failed Tasks, Server Load
[132]	Hybrid (Sim+Prac)	●	●	○	○	○	[143]	Sim	○	●	○	○	Finished Tasks
[138]	Analytic (Matlab)	○	●	○	○	○	[83]	Analytic (Matlab)	○	●	○	○	○
[42]	Sim	○	○	●	○	○	[117]	Prac	○	●	○	○	Bandwidth
[45]	Analytic	○	●	○	○	○	[90]	Sim	●	●	○	●	Resource Usage
[149]	Sim	○	●	○	○	○	[14]	Analytic (Matlab)	○	●	●	○	Failed Transmission
[53]	Sim	○	●	○	○	Weighted Cost	[58]	Analytic (Matlab)	○	●	○	○	○
[64]	Hybrid (Sim, Prac) (iFogSim)	●	●	●	●	○	[114]	Sim (Sumo)	○	●	○	○	○
[128]	Analytic	○	●	○	○	Average Offloaded Tasks	[6]	Sim	○	●	○	○	○
[33]	Analytic (Matlab)	○	●	●	○	○	[16]	Sim (Edge SimDAG)	○	●	●	○	Success rate, Utilization

[19]	Analytic	○	○	○	○	System Utility
[11]	Prac	○	●	○	○	○
[12]	Analytic	○	●	○	●	○
[21]	Prac	○	○	○	●	○
[74]	Hybrid (Sim, Prac)	○	○	○	●	○
[30]	Prac	○	●	○	○	○
[155]	Hybrid (Sim, Prac)	○	●	○	○	○
[135]	Sim	○	●	○	○	○
[110]	Hybrid (Sim, Prac)	○	○	●	○	○
[88]	Hybrid (Sim, Prac)	○	●	○	○	○
[142]	Analytic	●	●	●	○	○
[24]	Prac	○	●	○	○	Startup Time, Ram Usage
[49]	Analytic	○	○	○	○	Performance Gain
[145]	Sim (Cloudsim)	○	●	●	○	○
[57]	Analytic	○	○	○	○	Weighted Reward
[80]	Sim (iFogSim)	●	●	○	○	QoS
[94]	Sim (SimPy)	○	●	○	●	Application Loss
[124]	Analytic	○	○	○	●	Deadline Miss Ratio
[78]	Sim (iFogSim)	○	●	○	○	Deadline Miss Ratio
[22]	Sim	○	●	●	○	Weighted Cost
[4]	Analytic (Matlab)	○	●	○	○	Throughput
[50]	Analytic (Matlab)	○	●	●	○	Weighted Cost
[59]	Analytic (Matlab)	○	●	●	○	○
[61]	ND	○	●	○	○	Missed Deadline
[65]	Analytic (Matlab)	○	●	○	○	○
[69]	Analytic	○	●	○	○	Resource Utilization
[82]	Sim	○	●	○	●	Availability
[95]	Prac	○	●	●	●	Utilization
[101]	Sim (iFogSim)	●	●	○	○	Network Usage
[98]	Sim	○	○	○	○	Utility Function
[103]	Sim (iFogSim)	○	●	○	○	Network Usage
[104]	ND	○	●	●	○	System Gain
[108]	Prac	○	●	○	○	Deployed Instances
[115]	Sim	○	●	○	○	Weighted Cost
[116]	Analytic (Matlab)	○	●	○	○	Throughput
[134]	Sim	○	●	●	○	Weighted Cost
[46]	Sim	○	●	●	○	Weighted Cost
[102]	Sim (Fog WorkflowSim)	●	●	●	●	○

DO: Decision Overhead, T: Time, E: Energy, C: Monetary Cost, O: Other, Analytic: Analytical, Sim: Simulation, Prac: Practical, ●: Yes, ○: No

2. Besides, in a real-world scenario, application modules have different characteristics (e.g., computation size, input size, ram usage). Thus, the best assumption for application modules is applications with heterogeneous granularity specifications. As the number of contributing parameters and the dynamicity of the application elements increases, capturing the application parameters with temporal patterns for efficient scheduling decisions becomes more complex [39, 129]. Although traditional-based placement techniques (e.g., heuristic, meta-heuristic) often work well in general scenarios, they fail to adapt to continuous changes and dynamic contexts. ML-based decision engines, such as RL, can more efficiently work in a dynamic context and provide higher adaptability.

3. In large-scale Fog computing environments, numerous IoT applications with different workload models and hybrid CCR may exist. Hence, the decision engine requires an admission control component with an appropriate queuing mechanism (based on application requirements) to manage diverse incoming requests and prioritize them for making the decision.

4. Regarding the environmental architecture, the most generalized scenario is when the environment consists of several heterogeneous IoT devices, several heterogeneous FSs, and multiple heterogeneous CSs. Also, the required mechanisms for intra-tier and inter-tier cooperation among servers should be embedded to support diverse IoT application scenarios, such as mobility. Besides, multiple distributed servers can collaboratively provide better performance for the execution of IoT applications. Moreover, different fault domains can be prepared to improve the availability of services. However, as the number of IoT applications and available servers in the environment increase, the complexity of making decisions increases. Hence, the optimal scheduling decision cannot be obtained in a timely manner. Consequently, other placement techniques such as heuristics and ML-based techniques should be employed to obtain the scheduling decision in a reasonable time.

5. The decision engine can be implemented as a set of distributed services/microservices. A decision engine developed as a monolithic application may not be able to be deployed on a single server, especially on resource-limited FSs. Hence, distributed deployment of decision engine components on several distributed servers can provide several benefits: 1) more efficient deployment of resource-limited devices, 2) provides better fault tolerance 3) offers better scalability 4) support different deployment models (e.g., deployment of decision engine on FSs, CSs, or hybrid on both FSs and CSs). Hybrid deployment of decision engine components on both FSs and CSs can lead to a better user experience for end-users. To illustrate, applications requiring low latency and startup time can be managed at the low-level FSs (i.e., at the Edge), and then be scheduled based on the decision engine deployed at the Edge. However, application requests that are insensitive to latency or startup time can be forwarded to CSs for scheduling.

6. Regardless of application and environmental characteristics, failure recovery mechanisms and policies should be integrated into any decision engine. Independent failures and the non-deterministic nature of any components (either hardware or software) in distributed systems cause the most impactful issues in distributed systems. If the decision engine, which manages the scheduling and execution of incoming IoT application requests, does not have an appropriate failure recovery mechanism, the smooth execution of the whole system stalls.

9 FUTURE RESEARCH DIRECTIONS

This section presents future research directions, guiding researchers to further progress in the field of Fog computing.

Microservices-based applications. The popularity of microservices for the deployment of IoT applications is due to their loosely-coupled design, modularity, and the capability of microservices to be shared among multiple IoT applications. But, it may incur data consistency and data privacy challenges. To overcome these challenges, the placement techniques should consider the context of applications and data before sharing microservices.

Practical Container orchestration in Fog computing. Orchestrating container-based IoT applications is well studied in the cloud computing paradigm. However, in Fog computing, in which CSs and FSs collaborate to run an application, several deployment models of orchestration techniques are available. To illustrate, the master node can either be deployed on a FS or CS. When the master node runs on a FS, the communication overhead and latency for end-users will be reduced. However, the master node will use the most of resources on the FS for the cluster management, especially for resource-limited FSs. Also, when the master runs on a CS, the startup time and application latency will be negatively affected. Thus, based on the application structure and its goal, different container orchestration models should be studied to find the best deployment model according the application scenario. Several practical studies can be conducted to find which deployment model is suitable for each IoT application scenario in terms of communication overhead, the startup time of services, memory footprint, failure management, load balancing, and scheduling.

Hybrid scheduling decision engines. Usually, decision engines only use one placement technique for different IoT applications. However, the requirements of IoT applications are heterogeneous, where one application is sensitive to startup time while the extremely high accuracy is not important, or vice versa. Besides, decision engines should be adapted to work with either single or batch placement approaches. Hence, context-aware decision engines with a suite of placement techniques can be implemented to address the requirements of different IoT applications.

Systems for ML. Due to advancements in ML techniques and their rapid adoptions across many IoT applications, it creates new demand for specialized hardware resources and software frameworks (e.g., Nvidia GPU-powered Jetson, Google Coral Edge Tensor Processing Unit (Edge TPU)) for Fog computing. New systems and software frameworks should be built to support the massive computational requirement of these AI workloads. Besides, these systems can be a potential target for the deployment of decision engines due to their high computational capacity.

ML for systems. While ML systems themselves are becoming mature and adopted into many critical application domains, it is equally important to use these ML techniques to design and operate large-scale systems. Adopting the ML techniques to solve different resource management problems in Edge/Fog and Cloud is crucial to managing these complex infrastructures and workloads. Moreover, majority of ML techniques are not optimized to run on resource-constrained devices. To illustrate, consider an efficient ML model trained for resource management. Many resource-constrained devices require full integer quantization to run the trained model. However, post quantization of trained models is not always possible and in some cases they cannot be efficiently converted. As a result, a study on requirements for the efficient execution of resource management ML models on resource-limited FSs should be conducted.

Thermal management. The temperature of FSs (e.g., racks of Rpi or Nvidia Jetson platform), especially those executing large workloads, increases significantly. So, the cooling systems should be embedded to avoid system breakdown. Hence, a study on the temperature of these devices based on their main processing and communication modules can be conducted to find the respective temperature dynamics in different application scenarios and workloads. Moreover, lightweight thermal management software systems for FSs can be designed to control the temperature dynamics of devices. Also, the thermal index can be added as an important optimization/decision parameter alongside other currently available parameters (e.g., time, energy, cost) for the placement techniques.

Trade-off between execution cost of IoT devices and resource providers. The goal of scheduling algorithms is to minimize the execution cost of applications either from IoT or resource providers' perspectives. However, some parameters such as energy consumption or carbon footprint should be considered from both perspectives. Hence, not only is minimizing these parameters from either perspective critical to reducing total energy consumption, but a trade-off parameter between the execution cost of IoT devices and resource providers can be designed, aiming at total energy or carbon footprint minimization.

Privacy aware and adaptive decision engines. Data-driven and distributed scheduling approaches are gaining popularity due to their high adaptability and scalability. However, sharing raw data of users or systems incurs privacy issues. To illustrate, in DDRL-based scheduling techniques, sharing experiences of multiple agents significantly reduce the exploration costs and improve convergence time of DDRL agents while incurring privacy concerns when raw agents' experiences are shared. Accordingly, privacy-aware mechanisms for sharing such data (e.g., agents' experiences) can be integrated with these highly adaptive distributed scheduling techniques.

10 SUMMARY

In this paper, we mainly focused on scheduling IoT applications in Fog computing environments. We identified several main perspectives that play an important roles in scheduling IoT applications, namely application structure, environmental architecture, optimization characteristics, decision engines properties, and performance evaluation. Next, we separately identified and discussed the main elements of each perspective and provided a taxonomy and research gaps in the recent literature. Finally, we highlighted several future research directions for further improvement of Fog computing.

REFERENCES

[1] Business Insider. 2020. THE INTERNET OF THINGS 2020. https://www.businessinsider.com/internet-of-things-report. [Online; accessed 20-October-2021].

[2] IDC. 2020. IoT Growth Demands Rethink of Long-Term Storage Strategies. https://www.idc.com/getdoc.jsp?containerId=prAP46737220. [Online; accessed 20-October-2021].

[3] Mohammad Aazam, Sherali Zeadally, and Khaled A Harras. 2018. Offloading in fog computing for IoT: Review, enabling technologies, and research opportunities. *Future Generation Computer Systems* 87 (2018), 278–289.

[4] Mohamed Abd Elaziz, Laith Abualigah, and Ibrahim Attiya. 2021. Advanced optimization technique for scheduling IoT tasks in cloud-fog computing environments. *Future Generation Computer Systems* (2021).

[5] Mohamed Abdel-Basset, Reda Mohamed, Mohamed Elhoseny, Ali Kashif Bashir, Alireza Jolfaei, and Neeraj Kumar. 2020. Energy-aware marine predators algorithm for task scheduling in IoT-based fog computing applications. *IEEE Transactions on Industrial Informatics* 17, 7 (2020), 5068–5076.

[6] Raafat O Aburukba, Taha Landolsi, and Dalia Omer. 2021. A heuristic scheduling approach for fog-cloud computing environment with stationary IoT devices. *Journal of Network and Computer Applications* 180 (2021), 102994.

[7] Mainak Adhikari, Satish Narayana Srirama, and Tarachand Amgoth. 2021. A comprehensive survey on nature-inspired algorithms and their applications in edge computing: Challenges and future directions. *Software: Practice and Experience* (2021).

[8] Cosimo Anglano, Massimo Canonico, Paolo Castagno, Marco Guazzone, and Matteo Sereno. 2020. Profit-aware coalition formation in fog computing providers: A game-theoretic approach. *Concurrency and Computation: Practice and Experience* 32, 21 (2020), e5220.

[9] Alia Asheralieva and Dusit Niyato. 2021. Learning-based mobile edge computing resource management to support public blockchain networks. *IEEE Transactions on Mobile Computing* 20, 3 (2021), 1092–1109.

[10] Enzo Baccarelli, Paola G Vinueza Naranjo, Michele Scarpiniti, Mohammad Shojafar, and Jemal H Abawajy. 2017. Fog of everything: Energy-efficient networked computing architectures, research challenges, and a case study. *IEEE access* 5 (2017), 9882–9910.

[11] Hossein Badri, Tayebeh Bahreini, Daniel Grosu, and Kai Yang. 2020. Energy-aware application placement in mobile edge computing: A stochastic optimization approach. *IEEE Transactions on Parallel and Distributed Systems* 31, 4 (2020), 909–922.

[12] Tayebeh Bahreini, Hossein Badri, and Daniel Grosu. 2022. Mechanisms for resource allocation and pricing in mobile edge computing systems. *IEEE Transactions on Parallel and Distributed Systems* 33, 3 (2022), 667–682.

[13] Tayebeh Bahreini, Marco Brocanelli, and Daniel Grosu. 2021. VECMAN: A Framework for Energy-Aware Resource Management in Vehicular Edge Computing Systems. *IEEE Transactions on Mobile Computing* (2021). (in press).

[14] Hayat Bashir, Seonah Lee, and Kyong Hoon Kim. 2019. Resource allocation through logistic regression and multicriteria decision making method in IoT fog computing. *Transactions on Emerging Telecommunications Technologies* (2019), e3824.

[15] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. 2012. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 13–16.

[16] Lingfeng Cai, Xianglin Wei, Changyou Xing, Xia Zou, Guomin Zhang, and Xiulei Wang. 2021. Failure-resilient DAG task scheduling in edge computing. *Computer Networks* 198 (2021), 108361.

[17] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. 2011. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and experience* 41, 1 (2011), 23–50.

[18] Lixing Chen, Cong Shen, Pan Zhou, and Jie Xu. 2021. Collaborative service placement for edge computing in dense small cell networks. *IEEE Transactions on Mobile Computing* 20, 2 (2021), 377–390.

[19] Weiwei Chen, Dong Wang, and Keqin Li. 2019. Multi-user multi-task computation offloading in green mobile edge cloud computing. *IEEE Transactions on Services Computing* 12, 5 (2019), 726–738.

[20] Xianfu Chen, Honggang Zhang, Celimuge Wu, Shiwen Mao, Yusheng Ji, and Medhi Bennis. 2019. Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning. *IEEE Internet of Things Journal* 6, 3 (2019), 4005–4018.

[21] Yu Chen, Sheng Zhang, Yibo Jin, Zhuzhong Qian, Mingjun Xiao, Jidong Ge, and Sanglu Lu. 2022. LOCUS: User-Perceived Delay-Aware Service Placement and User Allocation in MEC Environment. *IEEE Transactions on Parallel and Distributed Systems* 33, 7 (2022), 1581–1592.

[22] Zhipeng Cheng, Minghui Min, Minghui Liwang, Lianfen Huang, and Zhibin Gao. 2021. Multi-Agent DDPG-Based Joint Task Partitioning and Power Control in Fog Computing Networks. *IEEE Internet of Things Journal* 9, 1 (2021), 104–116.

[23] A.V. Dastjerdi, H. Gupta, R.N. Calheiros, S.K. Ghosh, and R. Buyya. 2016. Fog Computing: principles, architectures, and applications. In *Internet of Things: Principles and Paradigms*, Rajkumar Buyya and Amir Vahid Dastjerdi (Eds.). Morgan Kaufmann, 61 – 75. https://doi.org/10.1016/B978-0-12-805395-9.00004-6

[24] Qifan Deng, Mohammad Goudarzi, and Rajkumar Buyya. 2021. FogBus2: a lightweight and distributed container-based framework for integration of IoT-enabled systems with edge and cloud computing. In *Proceedings of the International Workshop on Big Data in Emergent Distributed Environments*. 1–8.

[25] 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 Transactions on Mobile Computing* 20, 5 (2020), 1907–1923.

[26] Wanchun Dou, Wenda Tang, Bowen Liu, Xiaolong Xu, and Qiang Ni. 2020. Blockchain-based Mobility-aware Offloading mechanism for Fog computing services. *Computer Communications* 164 (2020), 261–273.

[27] Elie El Haber, Tri Minh Nguyen, Dariush Ebrahimi, and Chadi Assi. 2018. Computational cost and energy efficient task offloading in hierarchical edge-clouds. In *2018 IEEE 29th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*. IEEE, 1–6.

[28] Vajiheh Farhadi, Fidan Mehmeti, Ting He, Thomas F La Porta, Hana Khamfroush, Shiqiang Wang, Kevin S Chan, and Konstantinos Poularakis. 2021. Service placement and request scheduling for data-intensive applications in edge clouds. *IEEE/ACM Transactions on Networking* 29, 2 (2021), 779–792.

[29] Claudio Fiandrino, Nicholas Allio, Dzmitry Kliazovich, Paolo Giaccone, and Pascal Bouvry. 2019. Profiling performance of application partitioning for wearable devices in mobile cloud and fog computing. *Ieee access* 7 (2019), 12156–12166.

[30] Kaihua Fu, Wei Zhang, Quan Chen, Deze Zeng, and Minyi Guo. 2022. Adaptive Resource Efficient Microservice Deployment in Cloud-Edge Continuum. *IEEE Transactions on Parallel and Distributed Systems* 33, 8 (2022), 1825–1840.

[31] Pegah Gazori, Dadmehr Rahbari, and Mohsen Nickray. 2020. Saving time and cost on the scheduling of fog-based IoT applications using deep reinforcement learning approach. *Future Generation Computer Systems* 110 (2020), 1098–1115.

[32] Hend Kamal Gedawy, Karim Habak, Khaled Harras, and Mounir Hamdi. 2021. Ramos: A resource-aware multi-objective system for edge computing. *IEEE Transactions on Mobile Computing* 20, 8 (2021), 2654–2670.

[33] Sara Ghanavati, Jemal H Abawajy, and Davood Izadi. 2020. An energy aware task scheduling model using ant-mating optimization in fog computing environment. *IEEE Transactions on Services Computing* (2020). (in press).

[34] Mostafa Ghobaei-Arani, Alireza Souri, and Ali A Rahmanian. 2020. Resource management approaches in fog computing: a comprehensive review. *Journal of Grid Computing* 18, 1 (2020), 1–42.

[35] Mohammad Goudarzi, Qifan Deng, and Rajkumar Buyya. 2021. Resource Management in Edge and Fog Computing using FogBus2 Framework. *arXiv preprint arXiv:2108.00591* (2021).

[36] Mohammad Goudarzi, Zeinab Movahedi, and Masoud Nazari. 2016. Mobile cloud computing: a multisite computation offloading. In *Proceedings of the 8th International Symposium on Telecommunications (IST)*. IEEE, 660–665.

[37] Mohammad Goudarzi, Marimuthu Palaniswami, and Rajkumar Buyya. 2019. A fog-driven dynamic resource allocation technique in ultra dense femtocell networks. *Journal of Network and Computer Applications* 145 (2019), 102407.

[38] Mohammad Goudarzi, Marimuthu Palaniswami, and Rajkumar Buyya. 2021. A distributed application placement and migration management techniques for edge and fog computing environments. In *Proceedings of the 16th Conference on Computer Science and Intelligence Systems (FedCSIS)*. IEEE, 37–56.

[39] Mohammad Goudarzi, Marimuthu S Palaniswami, and Rajkumar Buyya. 2021. A Distributed Deep Reinforcement Learning Technique for Application Placement in Edge and Fog Computing Environments. *IEEE Transactions on Mobile Computing* (2021). (in press).

[40] Mohammad Goudarzi, Huaming Wu, Marimuthu Palaniswami, and Rajkumar Buyya. 2021. An application placement technique for concurrent IoT applications in edge and fog computing environments. *IEEE Transactions on Mobile Computing* 20, 4 (2021), 1298–1311.

[41] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. 2013. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future generation computer systems* 29, 7 (2013), 1645–1660.

[42] Fengxian Guo, Heli Zhang, Hong Ji, Xi Li, and Victor CM Leung. 2019. An efficient computation offloading management scheme in the densely deployed small cell networks with mobile edge computing. *IEEE/ACM Transactions on*

Networking 26, 6 (2019), 2651–2664.
[43] Harshit Gupta, Amir Vahid Dastjerdi, Soumya K Ghosh, and Rajkumar Buyya. 2017. iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments. *Software: Practice and Experience* 47, 9 (2017), 1275–1296.
[44] Pooyan Habibi, Mohammad Farhoudi, Sepehr Kazemian, Siavash Khorsandi, and Alberto Leon-Garcia. 2020. Fog computing: a comprehensive architectural survey. *IEEE Access* 8 (2020), 69105–69133.
[45] Zhenhua Han, Haisheng Tan, Xiang-Yang Li, Shaofeng H-C Jiang, Yupeng Li, and Francis CM Lau. 2019. Ondisc: Online latency-sensitive job dispatching and scheduling in heterogeneous edge-clouds. *IEEE/ACM Transactions on Networking* 27, 6 (2019), 2472–2485.
[46] Abhishek Hazra and Tarachand Amgoth. 2021. CeCO: Cost-efficient Computation Offloading of IoT Applications in Green Industrial Fog Networks. *IEEE Transactions on Industrial Informatics* (2021). (in press).
[47] Tai Manh Ho and Kim-Khoa Nguyen. 2020. Joint server selection, cooperative offloading and handover in multi-access edge computing wireless network: A deep reinforcement learning approach. *IEEE Transactions on Mobile Computing* (2020). (in press).
[48] Cheol-Ho Hong and Blesson Varghese. 2019. Resource management in fog/edge computing: a survey on architectures, infrastructure, and algorithms. *ACM Computing Surveys (CSUR)* 52, 5 (2019), 1–37.
[49] Zicong Hong, Wuhui Chen, Huawei Huang, Song Guo, and Zibin Zheng. 2019. Multi-hop cooperative computation offloading for industrial IoT–edge–cloud computing environments. *IEEE Transactions on Parallel and Distributed Systems* 30, 12 (2019), 2759–2774.
[50] Farooq Hoseiny, Sadoon Azizi, Mohammad Shojafar, Fardin Ahmadiazar, and Rahim Tafazolli. 2021. PGA: a priority-aware genetic algorithm for task scheduling in heterogeneous fog-cloud computing. In *Proceedings of the IEEE INFOCOM 2021-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 1–6.
[51] Miao Hu, Lei Zhuang, Di Wu, Yipeng Zhou, Xu Chen, and Liang Xiao. 2019. Learning driven computation offloading for asymmetrically informed edge computing. *IEEE Transactions on Parallel and Distributed Systems* 30, 8 (2019), 1802–1815.
[52] Pengfei Hu, Sahraoui Dhelim, Huansheng Ning, and Tie Qiu. 2017. Survey on fog computing: architecture, key technologies, applications and open issues. *Journal of Network and Computer Applications* 98 (2017), 27–42.
[53] Zheyuan Hu, Jianwei Niu, Tao Ren, Bin Dai, Qingfeng Li, Mingliang Xu, and Sajal K Das. 2021. An Efficient Online Computation Offloading Approach for Large-scale Mobile Edge Computing via Deep Reinforcement Learning. *IEEE Transactions on Services Computing* (2021). (in press).
[54] Liang Huang, Suzhi Bi, and Ying-Jun Angela Zhang. 2020. Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks. *IEEE Transactions on Mobile Computing* 19, 11 (2020), 2581–2593.
[55] Liang Huang, Xu Feng, Anqi Feng, Yupin Huang, and Li Ping Qian. 2018. Distributed deep learning-based offloading for mobile edge computing networks. *Mobile networks and applications* (2018), 1–8.
[56] Liang Huang, Xu Feng, Cheng Zhang, Liping Qian, and Yuan Wu. 2019. Deep reinforcement learning-based joint task offloading and bandwidth allocation for multi-user mobile edge computing. *Digital Communications and Networks* 5, 1 (2019), 10–17.
[57] Liang Huang, Xu Feng, Luxin Zhang, Liping Qian, and Yuan Wu. 2019. Multi-server multi-user multi-task computation offloading for mobile edge computing networks. *Sensors* 19, 6 (2019), 1446.
[58] Mohamed K Hussein and Mohamed H Mousa. 2020. Efficient task offloading for IoT-based applications in fog computing using ant colony optimization. *IEEE Access* 8 (2020), 37191–37201.
[59] Samia Ijaz, Ehsan Ullah Munir, Saima Gulzar Ahmad, M Mustafa Rafique, and Omer F Rana. 2021. Energy-makespan optimization of workflow scheduling in fog–cloud computing. *Computing* (2021), 1–27.
[60] Mir Salim Ul Islam, Ashok Kumar, and Yu-Chen Hu. 2021. Context-aware scheduling in Fog computing: A survey, taxonomy, challenges and future directions. *Journal of Network and Computer Applications* (2021), 103008.
[61] Chengen Ju, Yue Ma, Zhenyu Yin, and Feiqing Zhang. 2021. An Request Offloading and Scheduling Approach Base on Particle Swarm Optimization Algorithm in IoT-Fog Networks. In *Proceedings of the 13th International Conference on Communication Software and Networks (ICCSN)*. IEEE, 185–188.
[62] Vasileios Karagiannis and Stefan Schulte. 2020. Comparison of alternative architectures in fog computing. In *Proceedings of the 4th IEEE International Conference on Fog and Edge Computing (ICFEC)*. IEEE, 19–28.
[63] F Khodadadi, A V Dastjerdi, and R Buyya. 2016. Internet of things: an overview. *Internet of Things* (2016).
[64] Dragi Kimovski, Narges Mehran, Christopher Emanuel Kerth, and Radu Prodan. 2021. Mobility-Aware IoT Applications Placement in the Cloud Edge Continuum. *IEEE Transactions on Services Computing* (2021). (in press).
[65] Amit Kishor and Chinmay Chakarbarty. 2021. Task offloading in fog computing for using smart ant colony optimization. *Wireless Personal Communications* (2021), 1–22.

[66] Frank Alexander Kraemer, Anders Eivind Braten, Nattachart Tamkittikhun, and David Palma. 2017. Fog computing in healthcare–a review and discussion. *IEEE Access* 5 (2017), 9206–9222.

[67] Gilsoo Lee, Walid Saad, and Mehdi Bennis. 2019. An online optimization framework for distributed fog network formation with minimal latency. *IEEE Transactions on Wireless Communications* 18, 4 (2019), 2244–2258.

[68] Hai Lin, Sherali Zeadally, Zhihong Chen, Houda Labiod, and Lusheng Wang. 2020. A survey on computation offloading modeling for edge computing. *Journal of Network and Computer Applications* (2020), 102781.

[69] Bowen Liu, Xiaolong Xu, Lianyong Qi, Qiang Ni, and Wanchun Dou. 2021. Task scheduling with precedence and placement constraints for resource utilization improvement in multi-user MEC environment. *Journal of Systems Architecture* 114 (2021), 101970.

[70] Jiagang Liu, Ju Ren, Yongmin Zhang, Xuhong Peng, Yaoxue Zhang, and Yuanyuan Yang. 2021. Efficient Dependent Task Offloading for Multiple Applications in MEC-Cloud System. *IEEE Transactions on Mobile Computing* (2021). (in press).

[71] Zhaolin Liu, Xiaoxiang Wang, Dongyu Wang, Yanwen Lan, and Junxu Hou. 2019. Mobility-aware task offloading and migration schemes in SCNs with mobile edge computing. In *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 1–6.

[72] Haifeng Lu, Chunhua Gu, Fei Luo, Weichao Ding, and Xinping Liu. 2020. Optimization of lightweight task offloading strategy for mobile edge computing based on deep reinforcement learning. *Future Generation Computer Systems* 102 (2020), 847–861.

[73] Haodong Lu, Xiaoming He, Miao Du, Xiukai Ruan, Yanfei Sun, and Kun Wang. 2020. Edge QoE: Computation offloading with deep reinforcement learning for Internet of Things. *IEEE Internet of Things Journal* 7, 10 (2020), 9255–9265.

[74] Zhi Ma, Sheng Zhang, Zhiqi Chen, Tao Han, Zhuzhong Qian, Mingjun Xiao, Ning Chen, Jie Wu, and Sanglu Lu. 2022. Towards Revenue-Driven Multi-User Online Task Offloading in Edge Computing. *IEEE Transactions on Parallel and Distributed Systems* 33, 5 (2022), 1185–1198.

[75] Pavel Mach and Zdenek Becvar. 2017. Mobile edge computing: A survey on architecture and computation offloading. *IEEE Communications Surveys & Tutorials* 19, 3 (2017), 1628–1656.

[76] Henrik Madsen, Bernard Burtschy, G Albeanu, and FL Popentiu-Vladicescu. 2013. Reliability in the utility computing era: Towards reliable fog computing. In *Proceedings of the 20th International Conference on Systems, Signals and Image Processing (IWSSIP)*. IEEE, 43–46.

[77] Redowan Mahmud, Samodha Pallewatta, Mohammad Goudarzi, and Rajkumar Buyya. 2021. IFogSim2: An Extended iFogSim Simulator for Mobility, Clustering, and Microservice Management in Edge and Fog Computing Environments. *arXiv preprint arXiv:2109.05636* (2021).

[78] Redowan Mahmud, Kotagiri Ramamohanarao, and Rajkumar Buyya. 2018. Latency-aware application module management for fog computing environments. *ACM Transactions on Internet Technology (TOIT)* 19, 1 (2018), 1–21.

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

[80] Redowan Mahmud, Satish Narayana Srirama, Kotagiri Ramamohanarao, and Rajkumar Buyya. 2019. Quality of Experience (QoE)-aware placement of applications in Fog computing environments. *J. Parallel and Distrib. Comput.* 132 (2019), 190–203.

[81] Redowan Mahmud, Adel N Toosi, Kotagiri Ramamohanarao, and Rajkumar Buyya. 2020. Context-aware placement of Industry 4.0 applications in fog computing environments. *IEEE Transactions on Industrial Informatics* 16, 11 (2020), 7004–7013.

[82] Adyson M Maia, Yacine Ghamri-Doudane, Dario Vieira, and Miguel Franklin de Castro. 2021. An improved multi-objective genetic algorithm with heuristic initialization for service placement and load distribution in edge computing. *Computer Networks* 194 (2021), 108146.

[83] Prasenjit Maiti, Hemant Kumar Apat, Bibhudatta Sahoo, and Ashok Kumar Turuk. 2019. An effective approach of latency-aware fog smart gateways deployment for IoT services. *Internet of Things* 8 (2019), 100091.

[84] Erfan Farhangi Maleki, Lena Mashayekhy, and Seyed Morteza Nabavinejad. 2021. Mobility-Aware Computation Offloading in Edge Computing using Machine Learning. *IEEE Transactions on Mobile Computing* (2021). (in press).

[85] Yuyi Mao, Changsheng You, Jun Zhang, Kaibin Huang, and Khaled B Letaief. 2017. A survey on mobile edge computing: The communication perspective. *IEEE Communications Surveys & Tutorials* 19, 4 (2017), 2322–2358.

[86] Ismael Martinez, Abdelhakim Senhaji Hafid, and Abdallah Jarray. 2020. Design, Resource Management, and Evaluation of Fog Computing Systems: A Survey. *IEEE Internet of Things Journal* 8, 4 (2020), 2494–2516.

[87] Francesca Meneghello, Matteo Calore, Daniel Zucchetto, Michele Polese, and Andrea Zanella. 2019. IoT: Internet of threats? A survey of practical security vulnerabilities in real IoT devices. *IEEE Internet of Things Journal* 6, 5 (2019), 8182–8201.

[88] Jiaying Meng, Haisheng Tan, Xiang-Yang Li, Zhenhua Han, and Bojie Li. 2020. Online deadline-aware task dispatching and scheduling in edge computing. *IEEE Transactions on Parallel and Distributed Systems* 31, 6 (2020), 1270–1286.
[89] Minghui Min, Liang Xiao, Ye Chen, Peng Cheng, Di Wu, and Weihua Zhuang. 2019. Learning-based computation offloading for IoT devices with energy harvesting. *IEEE Transactions on Vehicular Technology* 68, 2 (2019), 1930–1941.
[90] Carla Mouradian, Somayeh Kianpisheh, Mohammad Abu-Lebdeh, Fereshteh Ebrahimnezhad, Narjes Tahghigh Jahromi, and Roch H Glitho. 2019. Application component placement in NFV-based hybrid cloud/fog systems with mobile fog nodes. *IEEE Journal on Selected Areas in Communications* 37, 5 (2019), 1130–1143.
[91] Carla Mouradian, Diala Naboulsi, Sami Yangui, Roch H Glitho, Monique J Morrow, and Paul A Polakos. 2017. A comprehensive survey on fog computing: State-of-the-art and research challenges. *IEEE communications surveys & tutorials* 20, 1 (2017), 416–464.
[92] Mithun Mukherjee, Lei Shu, and Di Wang. 2018. Survey of fog computing: Fundamental, network applications, and research challenges. *IEEE Communications Surveys & Tutorials* 20, 3 (2018), 1826–1857.
[93] 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.
[94] Yucen Nan, Wei Li, Wei Bao, Flavia C Delicato, Paulo F Pires, and Albert Y Zomaya. 2018. A dynamic tradeoff data processing framework for delay-sensitive applications in cloud of things systems. *J. Parallel and Distrib. Comput.* 112 (2018), 53–66.
[95] BV Natesha and Ram Mohana Reddy Guddeti. 2021. Adopting elitism-based Genetic Algorithm for minimizing multi-objective problems of IoT service placement in fog computing environment. *Journal of Network and Computer Applications* 178 (2021), 102972.
[96] Shubha Brata Nath, Harshit Gupta, Sandip Chakraborty, and Soumya K Ghosh. 2018. A survey of fog computing and communication: current researches and future directions. *arXiv preprint arXiv:1804.04365* (2018).
[97] José Leal D Neto, Se-Young Yu, Daniel F Macedo, José Marcos S Nogueira, Rami Langar, and Stefano Secci. 2018. ULOOF: A user level online offloading framework for mobile edge computing. *IEEE Transactions on Mobile Computing* 17, 11 (2018), 2660–2674.
[98] Thieu Nguyen, Thang Nguyen, Quoc-Hien Vu, Thi Thanh Binh Huynh, and Binh Minh Nguyen. 2021. Multi-objective Sparrow Search Optimization for Task Scheduling in Fog-Cloud-Blockchain Systems. In *Proceedings of the IEEE International Conference on Services Computing (SCC).* IEEE, 450–455.
[99] Opeyemi Osanaiye, Shuo Chen, Zheng Yan, Rongxing Lu, Kim-Kwang Raymond Choo, and Mqhele Dlodlo. 2017. From cloud to fog computing: A review and a conceptual live VM migration framework. *IEEE Access* 5 (2017), 8284–8300.
[100] Tao Ouyang, Zhi Zhou, and Xu Chen. 2018. Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing. *IEEE Journal on Selected Areas in Communications* 36, 10 (2018), 2333–2345.
[101] 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.
[102] Lei Pan, Xiao Liu, Zhaohong Jia, Jia Xu, and Xuejun Li. 2021. A Multi-objective Clustering Evolutionary Algorithm for Multi-workflow Computation Offloading in Mobile Edge Computing. *IEEE Transactions on Cloud Computing* (2021).
[103] Maycon Peixoto, Thiago Genez, and Luiz Fernando Bittencourt. 2021. Hierarchical Scheduling Mechanisms in Multi-Level Fog Computing. *IEEE Transactions on Services Computing* (2021). (in press).
[104] Guang Peng, Huaming Wu, Han Wu, and Katinka Wolter. 2021. Constrained Multi-objective Optimization for IoT-enabled Computation Offloading in Collaborative Edge and Cloud Computing. *IEEE Internet of Things Journal* 8, 17 (2021), 13723–13736.
[105] Charith Perera, Yongrui Qin, Julio C Estrella, Stephan Reiff-Marganiec, and Athanasios V Vasilakos. 2017. Fog computing for sustainable smart cities: A survey. *ACM Computing Surveys (CSUR)* 50, 3 (2017), 1–43.
[106] Euripides GM Petrakis, Stelios Sotiriadis, Theodoros Soultanopoulos, Pelagia Tsiachri Renta, Rajkumar Buyya, and Nik Bessis. 2018. Internet of things as a service (itaas): Challenges and solutions for management of sensor data on the cloud and the fog. *Internet of Things* 3 (2018), 156–174.
[107] Carlo Puliafito, Enzo Mingozzi, Francesco Longo, Antonio Puliafito, and Omer Rana. 2019. Fog computing for the internet of things: A survey. *ACM Transactions on Internet Technology (TOIT)* 19, 2 (2019), 1–41.
[108] Thomas Pusztai, Fabiana Rossi, and Schahram Dustdar. 2021. Pogonip: Scheduling asynchronous applications on the edge. In *Proceedings of the IEEE 14th International Conference on Cloud Computing (CLOUD).* IEEE, 660–670.
[109] Qi Qi, Jingyu Wang, Zhanyu Ma, Haifeng Sun, Yufei Cao, Lingxin Zhang, and Jianxin Liao. 2019. Knowledge-driven service offloading decision for vehicular edge computing: A deep reinforcement learning approach. *IEEE Transactions on Vehicular Technology* 68, 5 (2019), 4192–4203.
[110] Xiaoyu Qiu, Weikun Zhang, Wuhui Chen, and Zibin Zheng. 2021. Distributed and collective deep reinforcement learning for computation offloading: A practical perspective. *IEEE Transactions on Parallel and Distributed Systems* 32,

5 (2021), 1085–1101.
[111] Dadmehr Rahbari and Mohsen Nickray. 2020. Task offloading in mobile fog computing by classification and regression tree. *Peer-to-Peer Networking and Applications* 13, 1 (2020), 104–122.
[112] Rodrigo Roman, Javier Lopez, and Masahiro Mambo. 2018. Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges. *Future Generation Computer Systems* 78 (2018), 680–698.
[113] Farah Ait Salaht, Frédéric Desprez, and Adrien Lebre. 2020. An overview of service placement problem in fog and edge computing. *ACM Computing Surveys (CSUR)* 53, 3 (2020), 1–35.
[114] Hani Sami, Azzam Mourad, and Wassim El-Hajj. 2020. Vehicular-OBUs-as-on-demand-fogs: Resource and context aware deployment of containerized micro-services. *IEEE/ACM Transactions on Networking* 28, 2 (2020), 778–790.
[115] Hani Sami, Azzam Mourad, Hadi Otrok, and Jamal Bentahar. 2021. Demand-Driven Deep Reinforcement Learning for Scalable Fog and Service Placement. *IEEE Transactions on Services Computing* (2021). (in press).
[116] Indranil Sarkar, Mainak Adhikari, Neeraj Kumar, and Sanjay Kumar. 2021. A Collaborative Computational Offloading Strategy for Latency-sensitive Applications in Fog Networks. *IEEE Internet of Things Journal* (2021). (in press).
[117] Mennan Selimi, Llorenç Cerdà Alabern, Felix Freitag, Luís Veiga, Arjuna Sathiaseelan, and Jon Crowcroft. 2019. A lightweight service placement approach for community network micro-clouds. *Journal of Grid Computing* 17, 1 (2019), 169–189.
[118] Ali Shakarami, Mostafa Ghobaei-Arani, and Ali Shahidinejad. 2020. A survey on the computation offloading approaches in mobile edge computing: A machine learning-based perspective. *Computer Networks* (2020), 107496.
[119] Shashank Shekhar, Ajay Chhokra, Hongyang Sun, Aniruddha Gokhale, Abhishek Dubey, and Xenofon Koutsoukos. 2019. Urmila: A performance and mobility-aware fog/edge resource management middleware. In *Proceedings of the IEEE 22nd International Symposium on Real-Time Distributed Computing (ISORC)*. IEEE, 118–125.
[120] Jinfang Sheng, Jie Hu, Xiaoyu Teng, Bin Wang, and Xiaoxia Pan. 2019. Computation offloading strategy in mobile edge computing. *Information* 10, 6 (2019), 191.
[121] Syed Noorulhassan Shirazi, Antonios Gouglidis, Arsham Farshad, and David Hutchison. 2017. The extended cloud: Review and analysis of mobile edge computing and fog from a security and resilience perspective. *IEEE Journal on Selected Areas in Communications* 35, 11 (2017), 2586–2595.
[122] Jagdeep Singh, Parminder Singh, and Sukhpal Singh Gill. 2021. Fog computing: A taxonomy, systematic review, current trends and research challenges. *J. Parallel and Distrib. Comput.* (2021).
[123] Balázs Sonkoly, János Czentye, Márk Szalay, Balázs Németh, and László Toka. 2021. Survey on Placement Methods in the Edge and Beyond. *IEEE Communications Surveys & Tutorials* (2021).
[124] Georgios L Stavrinides and Helen D Karatza. 2019. A hybrid approach to scheduling real-time IoT workflows in fog and cloud environments. *Multimedia Tools and Applications* 78, 17 (2019), 24639–24655.
[125] Ming Tang and Vincent WS Wong. 2020. Deep reinforcement learning for task offloading in mobile edge computing systems. *IEEE Transactions on Mobile Computing* (2020). (in press).
[126] Koen Tange, Michele De Donno, Xenofon Fafoutis, and Nicola Dragoni. 2020. A systematic survey of industrial Internet of Things security: Requirements and fog computing opportunities. *IEEE Communications Surveys & Tutorials* 22, 4 (2020), 2489–2520.
[127] Ouyang Tao, Xu Chen, Zhi Zhou, Lirui Li, and Xin Tan. 2021. Adaptive User-managed Service Placement for Mobile Edge Computing via Contextual Multi-armed Bandit Learning. *IEEE Transactions on Mobile Computing* (2021). (in press).
[128] Shujuan Tian, Chang Chi, Saiqin Long, Sangyoon Oh, Zhetao Li, and Jun Long. 2021. User Preference-Based Hierarchical Offloading for Collaborative Cloud-Edge Computing. *IEEE Transactions on Services Computing* (2021). (in press).
[129] Shreshth Tuli, Shashikant Ilager, Kotagiri Ramamohanarao, and Rajkumar Buyya. 2022. Dynamic scheduling for stochastic edge-cloud computing environments using a3c learning and residual recurrent neural networks. *IEEE Transactions on Mobile Computing* 21, 3 (2022), 940–954.
[130] Shreshth Tuli, Redowan Mahmud, Shikhar Tuli, and Rajkumar Buyya. 2019. Fogbus: A blockchain-based lightweight framework for edge and fog computing. *Journal of Systems and Software* 154 (2019), 22–36.
[131] Kanupriya Verma, Ashok Kumar, Mir Salim Ul Islam, Tulika Kanwar, and Megha Bhushan. 2021. Rank based mobility-aware scheduling in Fog computing. *Informatics in Medicine Unlocked* (2021), 100619.
[132] Can Wang, Sheng Zhang, Zhuzhong Qian, Mingjun Xiao, Jie Wu, Baoliu Ye, and Sanglu Lu. 2020. Joint server assignment and resource management for edge-based mar system. *IEEE/ACM Transactions on Networking* 28, 5 (2020), 2378–2391.
[133] Dongyu Wang, Zhaolin Liu, Xiaoxiang Wang, and Yanwen Lan. 2019. Mobility-aware task offloading and migration schemes in fog computing networks. *IEEE Access* 7 (2019), 43356–43368.
[134] Jin Wang, Jia Hu, Geyong Min, Wenhan Zhan, Albert Zomaya, and Nektarios Georgalas. 2021. Dependent Task Offloading for Edge Computing based on Deep Reinforcement Learning. *IEEE Trans. Comput.* (2021). (in press).

[135] Jin Wang, Jia Hu, Geyong Min, Albert Y Zomaya, and Nektarios Georgalas. 2021. Fast adaptive task offloading in edge computing based on meta reinforcement learning. *IEEE Transactions on Parallel and Distributed Systems* 32, 1 (2021), 242–253.

[136] Lin Wang, Lei Jiao, Ting He, Jun Li, and Henri Bal. 2021. Service Placement for Collaborative Edge Applications. *IEEE/ACM Transactions on Networking* 29, 1 (2021), 34–47.

[137] Shangguang Wang, Yan Guo, Ning Zhang, Peng Yang, Ao Zhou, and Xuemin Sherman Shen. 2021. Delay-aware microservice coordination in mobile edge computing: A reinforcement learning approach. *IEEE Transactions on Mobile Computing* 20, 3 (2021), 939–951.

[138] Shiqiang Wang, Rahul Urgaonkar, Murtaza Zafer, Ting He, Kevin Chan, and Kin K Leung. 2019. Dynamic service migration in mobile edge computing based on Markov decision process. *IEEE/ACM Transactions on Networking* 27, 3 (2019), 1272–1288.

[139] Xiaofei Wang, Yiwen Han, Victor CM Leung, Dusit Niyato, Xueqiang Yan, and Xu Chen. 2020. Convergence of edge computing and deep learning: A comprehensive survey. *IEEE Communications Surveys & Tutorials* 22, 2 (2020), 869–904.

[140] Xiaojie Wang, Zhaolong Ning, Song Guo, and Lei Wang. 2020. Imitation learning enabled task scheduling for online vehicular edge computing. *IEEE Transactions on Mobile Computing* (2020). (in press).

[141] Zi Wang, Zhiwei Zhao, Geyong Min, Xinyuan Huang, Qiang Ni, and Rong Wang. 2018. User mobility aware task assignment for mobile edge computing. *Future Generation Computer Systems* 85 (2018), 1–8.

[142] Huaming Wu, William J Knottenbelt, and Katinka Wolter. 2019. An efficient application partitioning algorithm in mobile environments. *IEEE Transactions on Parallel and Distributed Systems* 30, 7 (2019), 1464–1480.

[143] Jindou Xie, Yunjian Jia, Zhengchuan Chen, and Liang Liang. 2019. Mobility-aware task parallel offloading for vehicle fog computing. In *Proceedings of the International Conference on Artificial Intelligence for Communications and Networks*. Springer, 367–379.

[144] Xiong Xiong, Kan Zheng, Lei Lei, and Lu Hou. 2020. Resource allocation based on deep reinforcement learning in IoT edge computing. *IEEE Journal on Selected Areas in Communications* 38, 6 (2020), 1133–1146.

[145] Xiaolong Xu, Qingxiang Liu, Yun Luo, Kai Peng, Xuyun Zhang, Shunmei Meng, and Lianyong Qi. 2019. A computation offloading method over big data for IoT-enabled cloud-edge computing. *Future Generation Computer Systems* 95 (2019), 522–533.

[146] Zun Yan, Peng Cheng, Zhuo Chen, Branka Vucetic, and Yonghui Li. 2021. Two-Dimensional Task Offloading for Mobile Networks: An Imitation Learning Framework. *IEEE/ACM Transactions on Networking* 29, 6 (2021), 2494–2507.

[147] Bo Yang, Xuelin Cao, Joshua Bassey, Xiangfang Li, and Lijun Qian. 2021. Computation offloading in multi-access edge computing: A multi-task learning approach. *IEEE transactions on mobile computing* 20, 9 (2021), 2581–2593.

[148] Chao Yang, Yi Liu, Xin Chen, Weifeng Zhong, and Shengli Xie. 2019. Efficient mobility-aware task offloading for vehicular edge computing networks. *IEEE Access* 7 (2019), 26652–26664.

[149] Lei Yang, Bo Liu, Jiannong Cao, Yuvraj Sahni, and Zhenyu Wang. 2021. Joint computation partitioning and resource allocation for latency sensitive applications in mobile edge clouds. *IEEE Transactions on Services Computing* 14, 5 (2021), 1439–1452.

[150] Jingjing Yao and Nirwan Ansari. 2018. QoS-aware fog resource provisioning and mobile device power control in IoT networks. *IEEE Transactions on Network and Service Management* 16, 1 (2018), 167–175.

[151] Ibrahim Yasser, Abeer Twakol, Abd El-Khalek, Ahmed Samrah, AA Salama, et al. 2020. COVID-X: novel health-fog framework based on neutrosophic classifier for confrontation covid-19. *Neutrosophic Sets and Systems* 35, 1 (2020), 1.

[152] 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. *Journal of Systems Architecture* 98 (2019), 289–330.

[153] Cheng Zhang and Zixuan Zheng. 2019. Task migration for mobile edge computing using deep reinforcement learning. *Future Generation Computer Systems* 96 (2019), 111–118.

[154] PeiYun Zhang, MengChu Zhou, and Giancarlo Fortino. 2018. Security and trust issues in Fog computing: A survey. *Future Generation Computer Systems* 88 (2018), 16–27.

[155] Gongming Zhao, Hongli Xu, Yangming Zhao, Chunming Qiao, and Liusheng Huang. 2021. Offloading Tasks With Dependency and Service Caching in Mobile Edge Computing. *IEEE Transactions on Parallel and Distributed Systems* 32, 11 (2021), 2777–2792.

[156] Ruiting Zhou, Xueying Zhang, Shixin Qin, John CS Lui, Zhi Zhou, Hao Huang, and Zongpeng Li. 2020. Online Task Offloading for 5G Small Cell Networks. *IEEE Transactions on Mobile Computing* (2020). (in press).

[157] Chao Zhu, Giancarlo Pastor, Yu Xiao, Yong Li, and Antti Ylae-Jaeaeski. 2018. Fog following me: Latency and quality balanced task allocation in vehicular fog computing. In *Proceedings of the 15th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*. IEEE, 1–9.