Characterizing application scheduling on edge, fog, and cloud computing resources

Prateeksha Varshney | Yogesh Simmhan

Department of Computational and Data Sciences, Indian Institute of Science, Bangalore, India

Correspondence
Yogesh Simmhan, Department of Computational and Data Sciences, Indian Institute of Science, Bangalore 560012, India.
Email: simmhan@iisc.ac.in

Summary
Cloud computing has grown to become a popular distributed computing service offered by commercial providers. More recently, edge and fog computing resources have emerged on the wide-area network as part of Internet of things (IoT) deployments. These three resource abstraction layers are complementary, and offer distinctive benefits. Scheduling applications on clouds has been an active area of research, with workflow and data flow models offering a flexible abstraction to specify applications for execution. However, the application programming and scheduling models for edge and fog are still maturing, and can benefit from learnings on cloud resources. At the same time, there is also value in using these resources cohesively for application execution. In this article, we offer a taxonomy of concepts essential for specifying and solving the problem of scheduling applications on edge, fog, and cloud computing resources. We first characterize the resource capabilities and limitations of these infrastructure and offer a taxonomy of application models, quality-of-service constraints and goals, and scheduling techniques, based on a literature review. We also tabulate key research prototypes and papers using this taxonomy. This survey benefits developers and researchers on these distributed resources in designing and categorizing their applications, selecting the relevant computing abstraction(s), and developing or selecting the appropriate scheduling algorithm. It also highlights gaps in literature where open problems remain.

KEYWORDS
application models, cloud computing, edge computing, fog computing, quality of service, scheduling algorithms

1 | INTRODUCTION

In computing, scheduling refers to the process of allocating computing resources to an application and mapping constituent components of that application onto those resources to meet certain quality of service (QoS) and resource conservation goals. The application itself may be represented in an abstract or concrete form using different programming primitives such as processes, threads, tasks, jobs, workflows, Petri nets, and so on. Similarly, the computing resource may be diverse, ranging from local cores and processors on a host, to distributed resources such as nodes in a cluster, virtual machines (VMs) in a cloud, edge or mobile devices in an Internet of things (IoT) deployment, or desktops in a volunteer computing network. QoS for the application, such as their latency, and conservation goals, such as minimizing
the quanta of resource or their energy footprint, can likewise be used to determine the schedule. Consequently, examining application scheduling requires us to understand the behavior of the computing resources, application models, and QoS goals in an integrated manner.

There is a growing availability of heterogeneous distributed computing resources. Cloud computing is a capability offered by commercial service providers using a rental model. Here, virtualized compute and storage resources at a large data center with thousands of servers are available on demand. In addition, there has been the emergence of devices offered by commercial service providers using a rental model. Here, virtualized compute and storage resources at a large city infrastructure, lifestyle gadgets such as wearables, and smart appliances or smartphones. In addition to sensing and generating observation streams, these devices that number in the tens of thousands have spare compute, storage, and memory capacities. These can be leveraged to execute IoT applications at the edge of the network. Further, there has been heightened interest in fog resources, which are between the edge and cloud in the network hierarchy, with compute, storage, and memory capacities that fall between these layers as well. These edge and fog resources provide the opportunity for low-latency processing of the generated data, closer to its source, and on the wide-area network. As a result, there is critical need to understand how this diverse ecosystem of edge, fog and cloud resources can be effectively used by large-scale distributed applications.

Scheduling applications on edge, fog, and cloud deals with the placement of the application’s logic components onto these resources for execution, deciding their interactions within this compute and network hierarchy, and managing various forms of dynamism, to meet their QoS requirements. There can be resource mobility at the edge and fog layers. The applications may also impose requirements on logical mobility of processes and data. Further, there may be changes in the data generation rates, network behavior, or energy levels of batteries that require reactive strategies. The application needs to be scheduled and coordinated to meet various QoS goals such as latency, energy, and monetary constraints. As a result, scheduling within this complex ecosystem involves a multitude of online coordination and optimization decisions, and the flow of control signals and data that can impact the application performance and resource usage.

There has been substantial work on examining scheduling approaches on clouds and clusters. However, given the nascency of edge and fog resources, there is a lack of a systematic review of distributing and scheduling applications on these resource classes individually, and together with clouds. Existing literature has proposed the conceptual foundations of edge and fog computing. Others, including us, have discussed the benefits and challenges involved in the coordination among edge, fog, and cloud layers in a hierarchical model, but have failed to examine in detail their impact of the applications and their schedule. Several scheduling approaches also do not distinguish between the edge and the fog layers, and subsume the former into the latter (or vice versa). There is also divergence in the assumptions made on the reliability and costing for the edge and fog layers. Hence, there is the need to understand the possible architectural patterns and scheduling mechanisms that have been proposed to cohesively schedule on these resource abstraction layers to inform researchers on open problems and developers on available approaches and their relative merits.

A literature survey can offer a framework to examine and understand such fast-paced emerging research, in the context of existing works. Preliminary surveys on mobile edge computing (MEC) review task off-loading strategies adopted by mobile edge devices that coexist with cloud resources, motivated by the growth in smartphones. However, these tend to focus on scheduling individual smartphone applications on a single edge device and the cloud. We generalize beyond mobile edges to all types of edge resource, include fog computing as a first-class entity, and consider diverse application and scheduling models on them. Some of these also consider the evolution of MEC with the advent of 5G communication technologies, with capabilities such as network slicing and network function virtualization. We approach this survey from the application and infrastructure-as-a-service (IaaS) perspectives, rather than examine the internals of the hardware or communications architectures. Others have also summarized the relative characteristics of the three resource layers, similar to the system design branch of our taxonomy. However, they do not examine the impact of this on application design and scheduling models. We distinguish our work from numerous cloud computing surveys, which review the breadth of the cloud ecosystem, scheduling of VMs onto hosts, and use of multicloud environments. These are at different levels of abstractions, even within cloud computing. We also contrast with specialized reviews on specific scheduling techniques or resource feature, such as metaheuristics, elasticity, and fault tolerance. Rather, we take a holistic view of these and other characteristics such as pricing, variable performance, and application models, and consider them in the presence of edge and fog resources as well, the application and system models as well. These related works are reviewed in greater detail in Section 5.

We address these gaps and present a survey on scheduling of applications on to edge, fog, and cloud computing resources, both independently and collectively, based on a review of contemporary scheduling literature. Specifically, we present a taxonomy of concepts and approaches for scheduling applications on edge, fog, and cloud resources (Section 3),
based on a detailed review of research literature over the past decade. This classification covers properties of edge, fog, and cloud resources relevant to scheduling, characteristics of the application that impacts the schedule, and the diverse QoS and constraints that determine the performance of the schedule, and categories of scheduling algorithms that exist. We then tabulate key scheduling literature using this taxonomy to offer a bird’s eye view of the landscape of application scheduling on these resources (Section 4). We place our survey in the context of other related surveys that exist and argue its novelty and impact (Section 5). Finally, we discuss emerging trends in this decade-old research area and highlight open problems that the research community is actively exploring at present (Section 6).

At the same time, our goal is not to examine specific implementations of edge or fog computing technologies and cloud service offerings beyond IaaS (and even that with an emphasis on computing resources) nor to offer case studies of applications. We compare and contrast the conceptual features across these resources and abstract the higher-order application models to help examine scheduling techniques. We also do not consider security and privacy aspects such as authentication, encryption, and cyberattacks on edge, fog, and cloud. Networking and communications technologies, and data center management are out of scope as well. Existing literature, some of which we review in the related work, address these adequately.

This article draws on our two prior works that characterize the resource behavior of edge, fog, and cloud22 and offer an overview of fog computing.13 These contents are selectively incorporated in the resource capabilities section (Section 3.1). However, the scope of this current review is substantially wider and more in depth, as is seen by the rest of the article.

In summary, our survey is based on the premise that: (1) it is important to consider edge, fog, and cloud resources collectively and in contrast to each other to leverage their mutual benefits; (2) this has to be examined from the application definition and scheduling perspective as it faces the end users and developers on these resources rather than the service providers; and (3) programming models and scheduling techniques on individual resource layers are translatable to others, in addition to exploring approaches that cut across these layers. To this end, we offer a novel and useful review of current literature and a consequent taxonomy related to these goals. As a result, it presents designers of scheduling algorithms for edge, fog, and cloud applications with a clear set of system and application features that they should consider for their target infrastructure. It also provides architects of application runtimes with the available options of scheduling algorithms that they can leverage to meet the needs of their end users. Also, it highlights gaps in existing literature where the intersection between these resource layers have yet to be adequately addressed.

2 | BACKGROUND

In this section, we provide a background of edge and fog computing to motivate the need to consider them as first-class computing resources, while substantiating this with a conceptual taxonomy for them later in Section 3.1. We then briefly discuss prior work on mobile cloud computing (MCC) (also called MEC), which has generalized into edge and cloud computing. This offers a contrast from this conceptual predecessor and scheduling strategies that have been attempted on it. Finally, we offer a similar distinction from the extensive work on scheduling for high-performance computing (HPC) resources, which is related to but has key distinctions from how applications are scheduled on the cloud. These establish clear contrasts from our effort while still offering a background on prior work on related technology domains.

2.1 | Edge and fog computing as an emerging resource abstraction

Edge computing refers to the use of thousands of computing devices such as sensors, gateways, mobile devices, or embedded systems at the edge of the network, often in the context of mobile phones or the IoT, for performing computation. This complements their traditional role of data collection and actuation, while the cloud is used for computation and data analytics.35,36 Fog computing,36 also known as cloudlets, was introduced by Satyanarayanan et al37, and popularized by Cisco as a complementary resource-rich layer that sits between the edge and the cloud.22 Fog provides data, compute, storage, and application services to end users similar to cloud data centers but with lower latency and faster response as it is typically one-hop away from the edge.38,39

There are many applications that motivate and benefit from edge and fog computing.22 There is a global push toward smart cities as a manifestation of IoT. Given the advances in deep learning, large-scale video surveillance has been adopted for public safety and as a proxy for ambient observations using analytics, such as identifying parking violations and classifying vehicles and people.40,41 Training the neural network models is computationally costly, and the source video streams at the edge are also large in size. Accelerated fog resources can complement edge resources available for deep
learning while reducing data transfer costs to the cloud. Smart power grids are another key domain in smart cities,\cite{22,42} with net-connected smart meters reporting power demand at households and industries every few minutes to the utility.\cite{43} Smart grid applications, such as demand-response optimization, help shape or shift power demand using forecasting models on the cloud that trigger curtailment strategies on the edge when a load mismatch is detected.\cite{44,45} State estimation to determine the health of the distribution network is even more time sensitive, $\mathcal{O}(ms)$, and needs computing at the edge.\cite{46}

While edge and fog computing are still emerging technologies, this taxonomy throws more light on these resource abstractions and their effective use from an application and scheduling model, based on current literature and technology.

### 2.2 Scheduling on mobile clouds versus edge computing

MCC (also called MEC or Mobile Clouds) is a precursor to the more general edge computing concept.\cite{47-49} It has a restricted design, motivated by cellular phones, both smart and feature phones, running applications or “apps.”\cite{50} MCC is concerned with strategies for offloading these applications from the mobile devices to the cloud due to the constrained computing power of the device. It involves a simple network topology and is often limited to direct communication between one mobile (edge) device and the cloud.

In the most common form of MCC, the coordination between the mobile device and the cloud is often on a per-application basis, i.e., each application is designed to run a part of its logic on the phone and the rest on the cloud. For example, the cloud may be used for data persistence, for looking up information, or for performing some costly computing tasks. The interaction may be using service endpoints, and this takes the form of a client-server architecture.

However, there exist research on more general frameworks that decide which applications or modules to offload, and when CloneCloud\cite{50} partitions a mobile application and migrates it to a device clone running in the cloud to minimize the application execution time and conserve the energy of the device. Barbera et al\cite{51} studied the trade-off between off-loading and not off-loading computation and software/data backup from mobile edge devices to the cloud, using bandwidth and energy consumption as metrics. Others plan of such off-loading at the cellular network level for different devices to the cloud, with apps defined as workflows.\cite{52}

MCC usually does not cooperatively schedule apps across a group of devices, due to security concerns of phones users, and energy constraints of the devices. Also, there are typically only the mobile device and the cloud layers. There is some literature on smartphones interacting with other nearby devices for performing computations,\cite{15} while others have also proposed using cellular towers as base stations and a fog-like layer. We generalize this even further by considering edge, fog, and cloud resource abstractions, and examining distributed scheduling across one or more of these. That said, MCC offers insights for such current efforts for scheduling and resource provisioning.

### 2.3 Scheduling on HPC clusters versus cloud computing

Traditionally, scheduling has been an important aspect for operating systems (OSs), HPC and supercomputing clusters, and computing grids.\cite{6,53} Grid computing offers shared distributed resources for which scheduling strategies are crucial, and these have been reviewed in detail.\cite{7,54} In contrast, cloud computing is a distributed computing capability offered by data center operators using a service-based model,\cite{11} while edge and fog computing operates on a wide-area network, and these affect how applications are scheduled upon them.\cite{55} We summarize key resource distinctions of HPC that impacts application design and scheduling, and motivates the need to separately explore scheduling on edge, fog, and cloud resources.

HPC centers traditionally have captive cluster infrastructure that are accessed by the center users using a batch queue that schedules jobs based on arrival time. The emphasis is on how best to allocate the available resources to the waiting jobs from tens to hundreds of users. In contrast, public cloud infrastructure offer access to VMs on demand without delay and give the illusion of “infinite” resources to thousands of users.\cite{56} It also allows the number of VMs requested to be elastically scaled up and down.\cite{57} At the same time, each application request tens of VMs rather than hundreds and thousands of cores common in HPC clusters.

Grids and HPC clusters use high-end fault-tolerant servers and high-performance networks to support floating point operations per second and numerical applications for communications. Clouds on the other hand use commodity and virtualized hardware, run on Ethernet, and are not as resilient to hardware failure. They also offer VMs of different resources capacities unlike HPC nodes that are typically homogeneous. As a consequence, HPC clusters can host tightly coupled large-scale applications with high throughput and reliability.\cite{58} Application makespan is the primary measure of success
for scheduling. Uniform nodes also limit the degrees of freedom when scheduling applications on such clusters. Clouds are popular for loosely coupled applications that run from seconds to days each and have variable resource needs. Fault tolerance is built in software due to weaker hardware robustness, and applications are more delay tolerant. Also, VMs may have variable performance due to virtualization and multitenancy. Factors such as VM acquisition lag and diverse VM sizes add to the scheduling complexity.

Finally, grids and HPC clusters encourage full use of their high-end infrastructure by their users to amortize the high capital costs. There is little or no financial cost to the users and at best quotas are imposed for fairness. As a result, scheduling algorithms prioritize the performance and makespan of the applications rather than conserve the resource usage. Public clouds follow a pay-as-you-go model with resources billed only for resources acquired on demand and diverse costing models. These offer scheduling algorithms a different parameter space to optimize upon.

### 3 | THE TAXONOMY

Our taxonomy takes a holistic view of application scheduling on edge, fog, and cloud resources, and offers a classification of the conceptual model of system models, application models, and their goals, which are required to design scheduling algorithms. We categorize the design of the scheduling algorithms and the approaches used to evaluate them. Specifically, the top levels of our taxonomy are: system design (Section 3.1), application model (Section 3.2), QoS goal (Section 3.3), and scheduling algorithm design (Section 3.4), as shown in Figure 1. A full view of the taxonomy is provided in in Figure A1 in the Appendix. In the following sections, we discuss each category in detail.

#### 3.1 | System design

Edge, fog, and cloud provide various types of computing resources with diverse capabilities and different pricing models. System design is concerned with the resource capacities, pricing features, and other system characteristics (Figure 2). For the cloud layer, we base our characterization on the capabilities of popular public IaaS clouds from Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Engine in defining these dimensions. For the edge and fog resources, these are based on current research and early commercial offerings such as Amazon Greengrass and Azure IoT Edge. We distill the essential and generic capabilities of such systems, and avoid transient capabilities offered in this fast-changing landscape.

#### 3.1.1 | Resource abstraction layer

In this section, we offer a relative overview of the three resource layers on which the applications are scheduled, as highlighted in Figure 5 and examined before. We have already introduced these resource classes, and now, we contrast their resource and performance characteristics. We base these on existing definitions, many of which place fog computing as a resource layer that fits between the edge devices and the cloud data centers, with features that resemble both.
The intrinsic distinction is the network distance between the edge/leaf of the Internet, where edge and fog resources are present and the core of the Internet where large cloud data centers are located. This affects the latency and available bandwidth between the different layers. This combined with where the data is generated, analytics are computed, decision signals are sent, and what QoS is required can affect the scheduling problem.

In addition, it is also worth considering the physical distance between the three computing paradigms and their accessibility by clients. In Figure 3A, four quadrants are formed from considering whether the resources within a layer are physically centralized or distributed (y-axis), and whether their access is global or restricted (x-axis). Resources in a cloud data center are centrally located, but depending on whether the cloud is public or private, are available to anyone in a pay-as-you-go model or only to users of the private corporation. That said, public cloud providers host geographically distributed data centers, sometimes several in a country or continent, while the number of large private data centers for an enterprise is more limited. Edge resources such as smartphones and set top boxes are distributed far and wide, but their access is restricted to individual users or managed applications. Fog resources are also physically distributed to be close to the edge but not as dispersed. Additional specializations, on whether there is a fog for each city block, one for the whole city or other variants, depend on the business models and applications that will evolve. One also expects the fog to offer as a shared, pay-as-you-go IaaS or platform-as-a-service (PaaS) model.

Edge and fog resources are distributed, which increases the probability of attacks and failures as discussed in Section 3.1.6. The access restrictions on private clouds and edge devices translate to a zone of trust for applications and services hosted on them, which enables sensitive data and services to be hosted on them. Fog and public clouds, however, are designed as shared resources with multitenancy, which require higher measures of security and sandboxing. That said, there may be fog architectures where the resources are deployed for specific applications or organization (e.g., a smart city municipality), similar to a private cloud. Further, the fog may sit at the boundary between public and private networks, and run proxy services that translate from one zone of trust to another, one service layer to another (e.g., constrained application protocol to hypertext transfer protocol), or one network protocol to another (e.g., IPv6 to IPv4).

It helps to understand the impact of mobility on these three resource layers, as illustrated in Figure 3B, as this impacts the communications, applications, and platform design. We distinguish between the mobility of the physical resource discussed here and the mobility of the logical applications, which we examine later. Cloud data centers, obviously, are not mobile, although their platforms can ease the mobility of data and applications among their data centers. Spatial mobility at edge devices is frequent, although not universal, e.g., mobility is seen in ubiquitous smartphones and autonomous vehicles, while they remain static in traffic cameras and smart power meters. This is more so in the context of MCC that is concerned with off-loading computations from mobile edge devices such as smartphones to cloud to save battery, speeding up computations, and creating data or software backup. Likewise, the fog layer can also manifest as a static or mobile resource. A fog server can be installed at fixed sites such as a coffee shop or the airport or on mobile vehicles such as taxi cabs or trains. This mobility can cause these resources to be unavailable, which is discussed later in Section 3.1.6.

Mobility at edge and fog layers also necessitate device discovery, as devices join, leave, and rejoin different parts of the network and resource fabric. This will on board and make them available as part of the resource pool and similarly remove them when they leave. The OpenFog Reference Architecture suggests a peer-to-peer (P2P) model where a new fog node broadcasts its information to a fog cluster. Others have proposed a publish-subscribe model for edge and fog resources.

![Figure 3](image-url)
devices to announce themselves on arrival or departure.\textsuperscript{21} This can be extended to a distributed hash table (DHT) as well, for notifying arrival and departure and for scheduling tasks.\textsuperscript{75} Some also suggest a hierarchical discovery approach for edges partitioned into fog parents, and fogs themselves reporting to higher-level fogs, all of which is accumulated at a discovery server.\textsuperscript{76} Some also use the transport-level protocols,\textsuperscript{77} eg, by having a leader device to broadcast an 802.11 WiFi beacon frame to notify spatially proximate devices that wish to join about the location of the leader to contact.\textsuperscript{78} Discovery using the software-defined networking layer is possible as well, as has been suggested for fog resources in a vehicular network.\textsuperscript{79} As a result, depending on the mobility of the edge and fog layers, the application and platform will need to be designed based on permanent, transient, periodic, or ephemeral connectivity between the layers and within the layers that can determine the reliability of access to data, storage, network, and computing resources.

The application definition needs to be scheduled and coordinated to meet various QoS goals such as latency, energy, and monetary constraints. This coordination can be done using different strategies, across edge, fog, and cloud resource layers. Three common orchestration models that are relevant in such a multilayered and distributed resource environment are centralized, hierarchical, and P2P. We also distinguish between scheduling decisions, the flow of control signals, and the flow of data, and different coordination models could be applied to these.

Centralized orchestration has a single service, either per application or for the platform, which is located in one of the three resource layers; makes scheduling decisions; and coordinates the transfer of control signals and/or data items.\textsuperscript{19} This is simple to design but can suffer from high latencies and transfer costs, and is a single point of failure. While this orchestrator often runs in the cloud (to coordinate across edge devices) or the edge (to interact with different cloud services), the fog layer could offer a sweet spot for such a centralized coordinator.\textsuperscript{80}

A hierarchical architecture is a generalization of the centralized model and allows only vertical communication of data and controls to take place between adjacent layers. This is a natural fit for fog computing as it leverages both the bandwidth and latency benefits of the fog layer in accelerating these flows, as well as the compute benefits closer to the observation source.\textsuperscript{17,19,39,40,81} Often, the cloud forms the root of this tree and is used for global data aggregation and coordination. Local data analytics is delegated to cloudlets and further to the edge devices. This allows a federated behavior that has shown to scale.

P2P is a form of distributed coordination that avoids a single point of failure. Here, peers in the same edge or fog layer can pass control and data directly among each other.\textsuperscript{68} The horizontal communication channels may initially be setup by an entity that has a global picture of the resources. This is typically done at the cloud or the fog or one of the edge devices that serves as a leader. There are simple component-based models for composing and executing P2P applications, as well as complex ones that use DHT to maintain an overlay network over peers that frequently enter and leave the system.\textsuperscript{14,71}

In a hybrid model, there are no strict limitations on the flow of control or data flows, and all layers are seen as having resources of heterogeneous characteristics. While there can be interconnections among resources within each layer (cloud, fog, and edge), communication can also take place vertically.\textsuperscript{72} This can require more complex coordination, but can potentially improve the resilience of the application when network connectivity between specific layers is interrupted.\textsuperscript{82}

### 3.1.2 Resource capacities and variability

Next, we consider the variability of the resource capacities offered on the different abstraction layers, as shown in Figure 4. Schedulers often leverage the ability to acquire resources of variable types and capacities as it allows a best fit between the application resource requirements and resources. Resources can vary in their types, sizes, and capacities across edge, fog, and cloud layers. This is illustrated in Figure 5 as inverted/upward pyramids that form a continuum across the layers, in increasing and decreasing order as applicable.

- **Homogeneous resources** Cloud computing resources are standardized within a service provider and are offered in different capacities based on a pay-as-you-go pricing model. IaaS providers offer VMs as their computing resource, characterized by their different compute capacities (CPU cores, clock speed, and architecture generation), physical memory, and

![FIGURE 4](wileyonlinelibrary.com) Taxonomy of resource variability [Colour figure can be viewed at wileyonlinelibrary.com]
network bandwidth, with associated pricing. These VM sizes are crucial in the context of scheduling cloud applications, and one can rely on the cloud provider to offer numerous homogeneous instances of a single resource size. Homogeneity may be possible in large-scale deployments of edge and fog, as a commercial service operation or by city utilities. Edge devices that are part of city-scale IoT infrastructure, such as net-connected smart power meters and traffic cameras for machine-to-machine interactions, may have uniform specifications.42 Fog resources such as Nvidia Jetson TX1 and Dell Edge Gateway can be also deployed as a standard across the city for commercial use, such as Barcelona’s “street-side cabinets”.83

- **Heterogeneous resources** At the cloud layer, a wide variety of VM sizes and configurations are offered, with AWS, eg, offering 42 different Elastic Compute Cloud (EC2) VM configurations. Besides resource capacities, these also vary in types of disks (solid-state drive or hard disk drive), presence of accelerators (GPUs), and higher-end architectures (DDR4 memory and latest CPU generation). While in the cloud, heterogeneity is a choice offered by the provider; on the edge and fog, this variability may be a necessity of the infrastructure deployment model. Heterogeneity is increased when the resources are consumer-owned instead of being available as part of commercial deployment, such as smartphones, smart watches, virtual reality headsets, etc. Edge platforms also tend to be constrained devices, with battery or memory capacity often being the limiting factor rather than even compute capability.22,84 The fog layer offers compute resources with a higher capacity than the edge but to a smaller scale than clouds.36,39 However, their resource capacity can vary,13 with Raspberry Pi devices at one end, to “micro (MDC)” or “nano” data centers on the other.20,85 The latter allow fogs to serve as a “reverse content delivery network” to let edge devices stage data on them and eventually push them to the cloud for archival, after some preprocessing.18,19,86 Resources may also have variable network characteristics.23 While the fog is close to the edge in the network topology, whether it is at a one-hop or multihop depends on the deployment.37 The fog is also expected to have a reliable and high-bandwidth Internet link,14 but its connectivity to the edge may be less robust due to the use of wireless links for the last mile.71

However, such diversity explores the dimensions that the scheduling algorithm has to examine. To mitigate this, some algorithms assume a uniform, homogeneous resource capacity (and price) for simplicity, which is more feasible on cloud resources.87 Algorithms may also limit themselves to leveraging just the compute diversity (eg, different VM or container sizes).17,63,88,89

### 3.1.3 Virtualization

Various types of virtualization (or lack of it) is possible within the different resource types, as shown by the taxonomy in Figure 6. Clouds expose every resource “as a service” using fabric software for infrastructure management, and this in part is a reason for its success.22,39,72 Virtualization using hypervisors offers two key benefits: (1) custom OS and software environments, and (2) sandboxing of VMs from each other using hardware-level support. The former allows user applications and Big Data platforms to work equally well on different clouds. They latter provides dependable resource allocation and security.

Resource-rich fog devices may be able to support hypervisors, and this can hasten their adoption for IaaS similar to clouds.22,72 On the other hand, resource-light fogs can use containers such as lxc and Docker, which offer users the control

---

**FIGURE 5** Resource characteristics of cloud, fog, and edge computing systems22 [Colour figure can be viewed at wileyonlinelibrary.com]

**FIGURE 6** Taxonomy of virtualization of resources [Colour figure can be viewed at wileyonlinelibrary.com]
over the software environment and limited resource sandboxing. However, security within multitenant containers on
a host is still a concern. That said, containers have limited memory overheads and rapid bootup time compared with
VMs, and this makes them preferred even for application management in the cloud. Containers launched within VMs is a
growing practice, with VMs offering the resource and security sandboxing and billing units, while containers (potentially
in tens in a VM) allow application environment management. Application management on the edge is a challenge, and
done in either ad hoc or tightly coupled to a platform like Android using sandboxed “apps.” Lastly, even these lightweight
wrappers may not be viable on severely constrained edge platforms. Hence, application may run on bare-metal servers or
within a PaaS (rather than IaaS).

There are also software fabrics that help manage such virtualized or containerized computing infrastructure, on the
cloud, fog, and edge. These serve as a form of a distributed OS. While public clouds such as Amazon, Microsoft, and Google
run their proprietary IaaS fabrics, OpenStack offers a full suite of compute, data, and network virtualization services for
private clouds. Kubernetes from Google focuses on container and compute management on large-scale clusters. Both
have also been extended to operate on fog and/or edge computing devices. While some have tried to use OpenStack as is on
fog resources, others extend its features to be customized for specific limitations of edge and fog resources. Lebre et al proposed a
decentralized P2P variant of OpenStack’s Nova compute service to enable wide-area computing resources, while Chang et al extended the quantum virtual network networking to support devices present on a network address
translation (NAT) network. OpenStack is also natively working on porting their capabilities to edge, fog, and MDC.
Similarly, Kubernetes has been used for compute containers on Raspberry Pi-class edge devices, and used to deploy
software on the fly on fog resources. Besides these, there are also open-source fabrics specialized for edge and fog, such
as Eclipse Kura and EdgeXFoundry that are evolving.

3.1.4 Pricing model

The monetary cost for accessing edge, fog, and cloud resources is variable. While clouds offer a mature pay-as-you-go
pricing model, edge and fog resources as yet have evolving business models. Typically, resource pricing is proportional to
its compute capacity on a given resource abstraction. However, there are also pricing differences due to the variable access
guarantees that are provided. One can broadly categorize the pricing model based on resources that are nonpreemptible and
those that preemptible (Figure 7). In the former, resources once provided are retained (and billed) until the application
relinquishes them. In the latter, resources may be taken back by the service provider at any time, and the application is
optionally compensated monetarily.

- **Nonpreemptible** These are the most common form of cloud resources where VMs have an associated fixed price per
  unit billing time based on their compute capacity. Clouds further distinguish this between on-demand VMs, which are
  acquired and released by the application flexibly, based on their current compute needs, and their billing intervals are as
  low as 1 s. Reserved VMs are those that are acquired in bulk for longer periods of time (e.g., one month) at a cheaper unit rate
  but billed in full irrespective of their usage. These are well-suited for users having a predictable long-term workload. The
diversity in cloud VM sizes also implies an associated diversity in the pricing of the resources. Variable-sized on-demand
VMs allow scheduling algorithms to make smart choices in trading off price to performance for their application.

Commercial fog providers may use consumption-based pricing where the users are billed as per their usage or
subscription-based pricing where the users pay a monthly fixed price and can use the fog resources networkwide, without
being pinned to a particular instance. These are similar to the on-demand and reserved models in clouds. The
infancy of fog deployments and their potential providers has implications on the operational costs as well. Alternatively,
smart cities may make them available as a utility service for free or based on payment. This may also extend
to edge devices that are part of the city’s deployment.
Preemptible Cloud providers may have spare capacity in their data center which are rented at a lower price than their on-demand counterparts, even 10× cheaper. This allows providers to increase the utilization and revenue of their data centers to offset the static operational overheads. While these VMs offer the same performance as a similarly sized on-demand VM, they are not guaranteed to be available for the user’s required duration and may be revoked by the provider. This requires scheduling algorithms to actively manage application checkpointing for reliability.

There are two models of such VMs available commercially. Amazon’s Spot VMs use an auction-based model that considers the highest price bids per billing interval for a VM size, with prices vary even within minutes. Amazon has recently started providing a 2-min warning when spot VMs are going to be revoked. Schedulers using such VMs must be aware of the current spot price and these revocation notices but can in turn reduce application execution costs by over 80%. Google and Microsoft offer preemptible VMs, which have a fixed discounted price that is significantly cheaper than their on-demand equivalents. This makes scheduling decisions easier than Amazon’s spot VMs with variable pricing. However, such VMs are currently limited to being used for a maximum of 24 hours. Google gives a 30-s warning before such VM are preempted.

Most edge devices and many noncommercial fog deployments are unreliable and transient, due to mobility, device uptime, and network connectivity issues. As a result, these resources have similarities to preemptible cloud resources where availability is not guaranteed. Here, the edge and fog resources are available opportunistically and often for free, with no assurance that one can acquire them at a given time or retain them for the required period. Applications may utilize the resources while they are in proximity but lose all progress if they go out of range or due to network unavailability. Mobile edge devices such as smartphones are one such example of opportunistic computation, and schedulers like Serendipity offload tasks to neighboring edge devices to minimize the execution time and save energy.

Hybrid: Scheduling strategies may take advantage of a mix of resources with different pricing schemes. One hybrid approach is to use captive resource capacity, such as reserved VMs or private clouds/clusters which are already paid for, along with on-demand resources that are paid as you go. Here, there are two possibilities: “Cloud bursting” or “Cloud firsting”. In the former, the scheduler gives priority to maximizing the use of the free captive resources before moving to on-demand (cloud) resources. In cloud firsting, the on-demand (cloud) resources are used by default and the limited captive resources used for instantaneous capacity access by the scheduler.

A second approach is to use a mix of on-demand and preemptible resources to reduce the cost of execution. Chohan et al use spot instances along with on-demand instances to speed up the MapReduce jobs on the cloud, while others propose strategies to manage the job’s life cycle on spot, on-demand, and captive resources. A third approach is to use resources from different resource abstraction layers that may have variable prices. While cloud and fog resources are reliable and available, they have higher costs as well. Opportunistic edge devices may be free but unreliable. So, when end users want the benefit of both cost reduction and reliability, scheduling algorithms may need to use more than one abstraction layer. Further, any free edge or fog devices may also play the role of captive or preemptible resource and mimic the first two approaches. For example, MCC uses unreliable but cheap mobile devices along with reliable clouds for offloading its computations, software, and data when the edge runs low on compute, storage, or battery.

3.1.5 Pricing characteristics

There are other pricing characteristics of resources besides the pricing model that should be considered, as shown in Figure 8.

Billing interval. In a pay-as-you-go model, users are charged per billing interval for which they use a resource. Cloud providers such as Amazon used to have a billing granularity of 60 min, where each whole or partial VM hour was billed as a full hour. However this has drop down to per-second billing over the last few years, sometimes with some minimum

![FIGURE 8 Taxonomy of pricing characteristics of resources](wileyonlinelibrary.com)
time, such as 10 min, that is charged. This has a consequence on the sizes of applications that are scheduled on VMs, the temporal granularity of control required by the scheduler, and the price paid. For example, in the work of Abrishami et al., the two different time intervals of 1 hour and 5 min are considered to evaluate the cost of scheduling, and as expected, the normalized cost is lower when considering the shorter billing interval. Billing models for edge and fog resources are still evolutionary, and there are not many commercial deployments that exist. They may undergo a similar pricing evolution as cloud over time.

- **Network pricing** While the pricing models above focus on compute resources, the cost of network bandwidth into and out of a resource layer may be charged, for example, for gigabytes of data transferred. This can be between edge and fog, fog and cloud, edge and cloud, or between resources in the same layer. Network pricing can either be symmetric or asymmetric. In the former, the price for both moving data in and out of a resource layer is charged equally, while in the latter, the bandwidth charges are different based on the direction. Often in cloud data centers, data-in and intradata center bandwidth are kept free to encourage hosting data in the cloud. These impact the costs for moving input/output data to/from the application on the cloud and the user's machine, as well as decisions regarding using captive off-cloud resources that require migrating application state over the network.

If edge and fog are deployed by the same provider or are a part of the same private network, such as a Wi-Fi access point, then there may be no pricing costs for the network usage. The data transfer within a layer and between these layers will be free. However, if edge and fog are part of different networks or the capacity of a constrained network is being saturated, then the data transfers may be chargeable. Also, connectivity among edge devices on different networks may be through gateways and proxies present on the fog or the cloud, to account for firewalls and network visibility. There can be additional charges for such redirection. Network pricing between edge or fog and the cloud layer is dependent on the deployment scenario. For example, edge/fog can be connected to the cloud with a broadband connection or 4G network which the local ISP may charge for.

- **Energy pricing** The energy profile can influence the capability and availability of some resources. Cloud data centers reduce their energy footprint but to limit operational costs. The fog is expected to run off grid power and, like the cloud, be energy conscious to lower pricing. However, there may be remote places where the fog runs on renewable energy such as solar, when energy-aware usage is a key goal. Edge devices are often concerned with battery life, and the choice of using specific edge features can depend on the current battery level. While resource providers usually include energy costs as part of the operational cost of a resource when pricing it, it may be possible to bill the energy costs separately based on the power consumed by an application. Alternatively, energy usage may be an application constraint or optimization specification when they run on edge and fog resources.

- **Price to performance ratio** Trade-off between resource performance and its price is important while selecting a resource. Clouds leverage economies of scale and usually have the lowest operational cost per resource unit. The elastic nature of VMs means that cloud providers attempt to raise prices linearly with the VM size, but the performance of larger VMs may be superlinear due to reduced network latency and resource contention on the same host. Scheduling algorithms can exploit these size differences, normalize price to performance, or leverage pricing arbitrage on spot VMs. Prices of edge and fog resources are dependent on the deployment scenario. The economies of scales will come into play if fog deployments are done at large scales. Consumer edge devices have lower capital costs due to their mass production, and zero operational costs if maintained by the user. However, fog/edge resources in the field, such as for IoT, may have higher operational costs due to maintenance and hence higher price to performance.

### 3.1.6 Other system characteristics

Besides resource pricing, types, and sizes, several other resource characteristics impact the design of scheduling algorithms (Figure 9), as we discuss below.

![Figure 9](https://wileyonlinelibrary.com)
Acquisition lag It might take some time for a resource to be ready for use after it is requested by a user. This delay varies with the type of resource requested and can affect the performance adversely if the scheduling algorithm frequently instantiates and switches between resources.\textsuperscript{87} Cloud VMs take tens of seconds or minutes to be provisioned, booted up, and ready for use by the end user, and this can vary with the number of VMs requested.\textsuperscript{61} Cloud scheduling algorithms may explicitly consider this lag in their planning.\textsuperscript{62,88} Edge or fog resources that run on bare-metal servers or on containers avoid the bootup time of hypervisor-based VMs but may not always have adequate on-demand capacity. This can cause queuing delays which add to the acquisition time. Some scheduling algorithms may be able to plan deferred acquisition (or advanced reservation) where a slot is assigned for the execution of a task on a resource at some later point of time and is available for the task with no lag at that time.

Performance variations Virtualized resources may not offer deterministic performance in terms of execution and data transfer time. These variations are often due to multitenancy where VMs collocated on the same host compete for resources or due to fabric management overheads.\textsuperscript{61} This can occur in containers too as the resource sandboxing is done by the OS and may not be as effective as hardware virtualization.\textsuperscript{115} These can cause the expected and observed makespans of the workflow to diverge by as much as 30\% and hence affect the performance of static schedules.\textsuperscript{64} Some scheduling algorithms consider performance variations as a first-class characteristic when allocating resources on the cloud.\textsuperscript{63,89,116-118}

Multitenancy Multitenancy allows different users to run their applications on the same host resource.\textsuperscript{13} The tenants may be separated from each other by VMs, containers, platforms, or not at all. Besides causing performance variations as mentioned above, these also impact the security and privacy of the applications and the data they process.\textsuperscript{68,84} Also, edge and fog devices may not be physically secured like a cloud data center, adding to the attack surface.\textsuperscript{20} Containerization offers less sandboxing between applications than virtualization, and data and applications in the fog and edge operate within a mix of trusted and untrusted zones.\textsuperscript{36,85} This can affect the scheduling of sensitive tasks in an application to be limited to trusted resources.

Failures Cloud resources are prone to occasional failures, which may happen due to disk and memory module failures, transient errors in network, cloud fabric and hypervisor failures, and power issues.\textsuperscript{119} These failures are rare, with commercial providers promising a monthly uptime of at least 99.95\%. However, such infrequent failures can still affect the execution of mission-critical applications adversely, with some algorithms addressing such situations. Clouds being more centralized are single points of failures but offer redundancy zones and alternate data centers. The wide-area-distributed nature of edge and fog resources increases their failure surface further.\textsuperscript{22} There is a higher chance of an edge device or fog server failing, their battery draining, or their network link dropping, and resiliency may need to be built into the scheduling strategy.\textsuperscript{82}

Availability Immediate availability of resources is a key feature of public clouds. While on-demand VMs have seemingly infinite availability, these are in practice limited to about 1000 VMs per customer.\textsuperscript{5,7} However, during periods of high demand, it is possible that a specific type of VM instance in a specific data center may not be available. Availability for preemptible VMs is naturally intermittent, while reserved instances offer guaranteed availability. Edge and fog resources may become unavailable due to transient network failures or a discharged battery but can be back online eventually (as opposed to a failure).\textsuperscript{15,22} This can cause a transient loss of access to data or compute on the edge or fog. Application running across all three resources may be affected by the weakest link.

3.2 Application model
A scheduling unit is the unit of application submitted by the user for resource allocation on the abstraction layers, along with QoS requirements. The application model we discuss here defines the constituent members of this scheduling unit, how it arrives, and when it is scheduled, as categorized in Figure 10. The scheduling algorithm may itself use coarser or finer granularity of scheduling. Also, the scheduling unit can have specific characteristics that help decide the resource mapping and techniques for fault tolerance.

3.2.1 Structure of scheduling unit
The execution environment offered by edge, fog, and cloud is inherently distributed, and the application space is vast.\textsuperscript{22} There are many application definitions in use, such as control flows, data flows, and event-driven models.\textsuperscript{120}

\footnote{https://aws.amazon.com/ec2/faqs/#How_many_instances_can_I_run_in_Amazon_EC2}
\footnote{https://azure.microsoft.com/en-in/documentation/articles/azure-subscription-service-limits}
Latency-sensitive applications may prefer an event-driven model that reacts rapidly to changing situations.\cite{40} Events streams are also lightweight.\cite{39} An application unit can have different structures (Figure 11).

- **Directed acyclic graph** Workflows, represented as a directed acyclic graph (DAG), are popular for capturing flow dependencies in complex distributed applications, and are widely used as a scheduling unit provided by the user for cloud, fog, and edge computing.\cite{15} A workflow DAG is a graph $G=(T,E)$, where $T$ is the set of task vertices, each of which form an atomic unit of scheduling, and $E$ is a set of control or data dependency edges between tasks. A single workflow can consist of one, a few, or thousands of tasks. Figure 13A shows a sample workflow with seven tasks and nine dependency edges. The number on each edge indicates the data size between the corresponding tasks, for example, in megabytes. The dependencies can be used to identify the order of execution of the tasks.

Workflows also offer additional information to guide their scheduling, such as the execution time for each task on a standard resource or on each resource size,\cite{56,63} or the number of instructions required by that task.\cite{89} The edges may be annotated with the size of the data transferred between the tasks to account for network transfer time and cost.\cite{59,114,116} The data flowing between tasks may be based on streams, microbatches, or files. The ability to define specialized data structures, compression, and transport mechanisms for distinct stream types such as video may be necessary too. The state associated with a task offers a context for execution and needs to migrate across resources.\cite{22,72} Tasks may also use location awareness or resource context in determining actions.\cite{73}

- **Task** An application may be monolithic, specified as a single task. Such a task degenerates to a singleton DAG. There are many application scheduling algorithms that limit themselves to scheduling just tasks rather than DAGs.\cite{88,105,121} For the purposes of the scheduler, a task structure is an opaque logic unit that is the smallest atomic unit of scheduling with no other task dependencies.

### 3.2.2 Mode of submission

Besides the unit of scheduling provided by the user, schedulers themselves may schedule individual or multiple units at a time, as shown in Figure 12. A user may provide a single scheduling unit (e.g., DAG and task), whose QoS requirements are independently specified and the scheduler schedules just this single unit. On the other hand, users may provide a set or series of units, with associated requirements, and the scheduler needs to meet the QoS across these multiple scheduling units. These can further be classified as homogeneous, where all the units have the same structure, as illustrated in Figures 13B and 13C, or heterogeneous where units can have a mix of DAGs and tasks, as seen in Figure 13D. This distinction helps in generating more efficient schedules. Heterogeneous scheduling units, while intuitive, are less common in existing literature.\cite{122} For example, consider both tasks of DAGs and individual tasks while generating mapping between the tasks and VMs.

Further, when multiple units are submitted for scheduling, all units may arrive at once as a batch or may arrive over time, in a transactional model, as shown in Figure 14. This interval of submission decides the information available to the scheduling algorithm. For example, in a batch mode, the resource needs and QoS for all units can be used to decide a “globally optimal” schedule. However, if the units arrive continuously, the scheduler may take individual decisions that
FIGURE 12  Taxonomy of number of scheduling units and their modes of submission [Colour figure can be viewed at wileyonlinelibrary.com]

FIGURE 13  Types of scheduling units. Figures 13B and 13C are examples of homogeneous scheduling units. Figure 13D is an example of heterogeneous scheduling units. A, A sample workflow DAG; B, A sample bag of tasks with 35 tasks; C, A sample bag of DAGs with five DAGs; D, A sample bag of tasks and DAGs [Colour figure can be viewed at wileyonlinelibrary.com]

FIGURE 14  The top illustrates a user submitting ten scheduling units in a batch mode, while the bottom shows submission of ten scheduling units in a transactional mode. A, Batch mode of submission; B, Transactional mode of submission
can affect the effectiveness of the schedule of future units. Typically, a batch has a shared QoS requirement defined on it, rather than individual QoS for each unit within. A batch may also be called a bag if there is no specific order in which the units need to be processed. Ensembles are a special type of homogeneous batch where all DAGs have a similar or the same structure, but different parameters (e.g., a parameter sweep), number of tasks, or task sizes.123,124 A bag of tasks (BoT) consists of a homogeneous batch of tasks that can be executed in any order. BoTs can achieve a high degree of task parallelism and efficiency across VMs and devices since they do not have interdependencies.113,121,125

A transactional mode is common when multiple users share the same application deployed on a resource (e.g., as a service (FAAS)), or a stream of microbatches arrive from an input source for processing by the application. Since requests arrive continuously, future workloads are hard to predict, and resources need to be dynamically allocated and deallocated.12,62,101,126

### 3.2.3 Granularity of scheduling

The taxonomy for granularity of scheduling is shown in Figure 15, and it identifies the various ways in which an algorithm processes the scheduling units after the user submits them. A single unit of scheduling means that the schedule is generated for one unit without considering other units which may be present, while in the case of a collection granularity, a schedule is generated for a collection of units as a whole, considering the impact of all units within the collection. This granularity of scheduling is related to the mode of submission. The natural choice is to schedule transaction mode of submissions as single units as they arrive and batch mode of submissions as collections.62,113,127

That said, it is possible for a scheduler to buffer units in a transactional workload for a certain time and generate a schedule for the collection, thereby increasing the resources efficiency, albeit at a higher latency.101,128 Similarly, if units within a batch do not have any collective QoS specified on them, the scheduler can “flatten” these units and consider them individually for scheduling. For example, in the work of Xu et al.,128 workflows are submitted by the users transactionally, and the ready tasks from the workflows are stored into an ordered queue where the tasks are sorted according to some rules. In the work of Skarlat et al.,24 IoT applications arrive at any time but the scheduling is done periodically, for applications which are closer to their deadlines.

### 3.2.4 Scheduling unit characteristics

We identify other characteristics of the scheduling unit that can impact the scheduling strategies, particularly under dynamic or failure conditions (Figure 16).

- **Resubmission** This allows a task (either individually or as part of a DAG) that has failed to be reexecuted completely from the start, without any side effects. This statelessness or idempotence property is minimally required to ensure fault tolerance of tasks on unreliable resources.129
- **Replication** This feature allows multiple copies of a task to be run simultaneously on different resources, without any side effect. This can enhance the guarantees for timely completion of a task even if one of the copies fail due to resource loss. This is particularly valuable for preemptible cloud resources or transient or unreliable edge or fog. It can also be used to opportunistically replicate a task on spare (free) resources so that the first to complete wins and can address resource underperformance.

- **Checkpointing** This allows the scheduler to save the state for a partially executed task and to resume it from the latest checkpoint to meet deadline constraints. This can be also useful when switching resources to improve the cost or time for execution. Checkpointing may require migration of the state to a persistent storage (e.g., a cloud storage service) or a different reliable resource before resumption, since the original device or VM may be transient. Checkpointing is leveraged when scheduling on preemptible VMs. On transient edge devices, CloneCloud uses trigger points to snapshot and migrate local state from the edge to its clone on the cloud to resume execution. Bittencourt et al. migrated the user's data from one cloudlet to another based on the user mobility in order to minimize latency.

The frequency of checkpointing is important. It balances the progress lost after the last checkpoint when a resource fails with the time and cost overhead for taking a checkpoint. Authors have used hourly or user-defined periods, the current and expected spot prices, the slack time allowed, the mobility and connectivity loss, and the job's progress to decide this. Failure handling can be done at task, DAG, and bag/ensemble level. However, not all techniques can be used at all levels. For example, while a workflow may be resubmitted upon failure, it may not be possible to resubmit an entire bag or ensemble. Also, while some critical tasks in a DAG may be replicated, it may be impractical to replicate the entire workflow.

- **Predictable runtime** The execution time of a task or DAG is deterministic if there is no uncertainty in its expected runtime on different resource sizes. In real-world scenarios, the actual runtime might vary due to performance variations and acquisition lag on edge, fog, and cloud resources (Section 3.1.6). The execution times may also change based on specific input parameters. These uncertainties affect the performance of scheduling algorithms. While many scheduling algorithms rely on the accurate runtime for scheduling units being available, others use initial estimates for the workflow tasks but later use real-time monitoring to revise the execution time and replan the schedule dynamically.

- **Data transfer time** Workflows and tasks may have large input and output data parameters that need to be transferred between tasks, within a DAG, or between a persistent storage location and the input/output interface to a single task or a DAG. These transfers can consume time and cost, both when moved between tasks on different resources in one layer, or between resources on different layers, using local- or wide-area networks. The application may specify these transfer sizes as part of its definition, and scheduling algorithms may consider them if provided. Data movement also requires storage and network to be available. In the work of Calheiros and Buyya, VMs are prematurely started to allow time to transfer in the data for a task scheduled on it. Others include the data transfer time to calculate the length of the critical path in a DAG. Some applications that pin the tasks on specific layers, such as the data preprocessing on the edge and analytics on the fog and/or cloud may force additional network latencies.

### 3.3 Quality-of-service goals

The goal of the scheduling algorithm is to determine a schedule that meets specific QoS requirements for the given scheduling unit. The QoS is characterized by the type of constraint that is imposed, the guarantees necessary in achieving the constraints, the optimization goal to evaluate the performance, and the scheduling granularity at which these requirements have to be met, as shown in the taxonomy tree in Figure 17. Besides these, quality of experience has been proposed as an alternative user-centric metric to QoS. It considers the user requirements and perceptions for a service in a particular context and calculated using prediction-based or feedback-based approaches. As such, this is a recent evolution and not considered in our review.
3.3.1 QoS constraint

Constraints indicate that the generated schedule must meet the constraint specification, while an optimization goal determines the performance of the schedule, provided the constraints are met. Time, cost, energy, and robustness are common metrics that are used as constraints and optimization goals, as shown in Figure 18.

- **Time** Temporal constraints are specified on the makespan of the scheduling unit, which is the time between it being submitted and it completing execution. Makespan includes any queue waiting time and transfer time, in addition to the actual task execution time. Users can require that the scheduling unit should complete its execution within a given deadline from the submission, based on its importance. Related to makespan is the concept of throughput, where the number of scheduling units executed per second is the metric. This is relevant for transactional mode of submission where the current or a target rate of requests must be supported.

- **Cost** For pay-as-you-go resources, the monetary cost is a key factor, and users may specify a budget constraint to bound the cost for running the scheduling unit.Schedulers may migrate applications to reliable but costlier resources when a deadline is imminent, but end up overpaying. So, rather than specify either or both of these constraints independently, users may include a utility function that combines both time and cost into a single dimension. The QoS specification can require that this given utility function fall within a certain bound.\(^{56,134,135}\)

- **Energy** The overall power consumption by the execution may be specified as a constraint. This may be important for edge devices whose cost may be free but have a limited battery life that should not be exhausted. The energy use may be due to both compute and communication. Edge or fog resources on the field may also run on renewable energy like solar, which have recharge cycles as well that factor into the energy constraint when scheduling.\(^{12}\)

- **Spatial** Geospatial constraints may be imposed on applications that must be processed on a device close to where the data are generated, eg, on the same device, the same private network, or some geofenced region, to ensure privacy and comply with security policies.\(^{81}\) While the physical security and network access are concerns on edge and fog, the geographical location and legal jurisdiction are factors on the cloud.\(^{22,66}\) These are complementary to resource locality required due to performance. Some applications may also pin specific tasks to specific resources, for physical access to on-board sensors or for access to a specific user or device context.\(^{40,72,73}\)

- **Robustness** Mission-critical applications may require guaranteed completion. This may pose a higher burden than just completing within a deadline and may require that failures not happen at all, rather than just be able to recover from failures within the deadline. Examples include power grid management or traffic signaling.\(^{39,81}\)

3.3.2 Constraint guarantee

The constraint specified by the user can be hard or soft (Figure 19). A hard constraint is inviolable, and its failure is catastrophic for the end user. Jin et al\(^{41}\) referred to applications with hard time constraints as *inelastic applications*, and they require real-time processing, typically to meet safety of humans, such as in autonomous vehicles.\(^{36}\) Soft requirements, on the other hand, need not strictly be achieved and a best effort is sufficient.\(^{62}\) In such cases, penalty functions may be used when the constraints are not satisfied. Such *elastic applications* (different from cloud elasticity) have flexibility in latencies and can perform batch processing, eg, for analyzing surveys from drones.\(^{22,73}\) In scheduling literature, the autoscaling mechanism used by Mao and Humphrey\(^{62}\) considers soft deadlines for the workflows while a reusable
3.3.3 Optimization goal

The QoS optimization goal attempts to minimize or maximize an objective function for the application’s schedule, as shown in Figure 20. The metric for the objective function is similar to the ones for the QoS constraint, such as the makespan, throughput, monetary cost, energy, utilization, faults, or a functional combination of these.

- **Minimization** Typically, the makespan, the cost or energy consumption, or the number of task failures are used as the objective function when minimizing it. A deadline constrained application may attempt to minimize the cost, with a number of such scheduling algorithms available.\(^\text{56,62,88,101,110,116}\) Similarly, under a budget constraint, the makespan may be the minimization goal.\(^\text{113,114}\) Others may lack a constraint and only aim to minimize, for example, the makespan, the energy consumption, or both time and cost.\(^\text{12,15,17,25,50,59,122,136}\)

- **Maximization** Common objective metrics used when maximizing include the application throughput and the utilization of resources. Skarlat et al\(^\text{23,24}\) maximized the utilization of cheap fog resources by trying to place more applications on it instead of the cloud to reduce the monetary cost. Wang et al\(^\text{137}\) scheduled BoTs to maximize the survivability given a deadline constraint, with task priorities used as a proxy for user-specific maximization goals. Malawski et al\(^\text{87}\) maximized the number of high-priority workflows from an ensemble that complete, given a fixed budget and deadline constraint.

- **Neither** It is also possible that no optimization goal is specified. In the work of Calheiros and Buyya,\(^\text{63}\) the user specifies a preferred deadline and a variable budget, and the goal is to meet the soft deadlines of workflows. Kushwaha and Simmhan\(^\text{104}\) analyzed the trade-offs between cost savings over fixed price VMs and resilience when tasks are run on spot VMs. In the work done by Brogi and Forti,\(^\text{66}\) users specify the hardware, software, bandwidth, and latency requirements for tasks and aim to enumerate all valid deployments on the fog and cloud.

3.3.4 Granularity

The QoS optimization goals and constraints can be specified at various granularities, with the default being at the granularity of the single scheduling unit or the batch, depending on the mode of submission. However, other variations in specifying the granularity of constraints and optimizations exist as well, as shown in Figure 21.

- **Directed acyclic graph** Placing the QoS on the DAG means that it has to be met for the DAG as a whole without regard to the QoS for individual tasks. In such cases, the constraints and the optimization goals are specified at the same granularity.\(^\text{56,110,116}\) However, these may be different as well. For example, the makespan constraint may be specified at the DAG level but the goal of minimizing cost is specified for a batch of DAGs.\(^\text{24,62,101}\) When constraints are specified for a DAG, the tasks lying on its critical path are essential to manage the makespan of the workflow. Scheduling algorithms may estimate and assign subdeadlines for each task to decide their mapping to a suitable resource, with tasks in the critical path having the least flexibility. For example, the algorithm called IaaS cloud partial critical paths with deadline distribution distributes the subdeadline of a path in a DAG to each task in the path, in proportion to its minimum execution time\(^\text{56}\) while the work by Liu et al\(^\text{138}\) uses the average execution time. Subdeadlines can be used to select the best resource that can...
execute the task within its subdeadline. In Serendipity, subdeadlines on each task are assigned to minimize the overall time and the energy consumed for executing the DAGs.

- **Batch** QoS can also be specified on a batch of scheduling units, be they a BoTs or an ensemble of workflows. The constraints and/or goals have to be met for the entire collection, without regard to individual units. For example, Varshney and Simmhan defined a deadline constraint and a cost minimization goal to schedule a BoTs executing on preemptible and on-demand VMs. It is possible that the scheduler may not compute the subconstraints for the individual units of the batch. In the work done by Deng et al., a delay constraint is specified for the batch of task, and the goal is to minimize the overall power consumption on fog resources. Skarlat et al periodically scheduled a batch of tasks with the goals of minimizing the makespan for the batch and maximizing the resource utilization.

- **Task** The user may also specify goals or constraints for individual tasks, be they tasks of a DAG, tasks within a bag, or single tasks. It is also possible for users to indicate the constraints and optimization goals individually for different stages of the DAG. In such cases, different algorithms can be used to schedule the tasks of different stages according to the constraints and goal at each stage.139

### 3.4 Scheduling algorithms

Given the edge, fog, and cloud resource environment, the application model, and QoS requirements, the scheduling algorithm is designed to schedule the application onto the resources to meet the QoS goal and meet the constraints. Solving this scheduling problem is computationally complex for nontrivial problem sizes. As a result, there is a large body of literature on techniques and algorithms to solve this problem, often to get an approximate rather than an optimal solution. Existing surveys classify scheduling algorithms based on various techniques that they employ, and these determine the quality of the schedule that is generated and the time taken to generate the same.8,31,34,140,141

As such, there are no intrinsic reasons why these prior classifications, as illustrated in Figure 22, are not equally applicable to application scheduling on edge, fog and clouds. However, having multiple resource layers across a wide-area network also brings in the opportunity for delegating the scheduling hierarchically (eg, cloud delegating a subset of the scheduling unit to a fog for scheduling on itself and its neighboring edges), or to make federated decisions, rather than just a centralized decision. We examine these and, to be holistic, also summarize outcomes from prior taxonomies in this section; we refer readers to these external sources for a detailed review of the algorithmic strategies.

#### 3.4.1 Scheduling techniques

Finding the optimal schedule is a combinatorial optimization problem and a variation of the classic “job shop scheduling” problem, the solution to which is NP hard.142 Brute force algorithms try all possible mappings of the scheduling units to the compute resources to arrive at a globally optimal solution while meeting the constraints. For example, a simple case of optimally placing M tasks to R resources has a brute-force computational complexity of \( O(R^M) \). This is intractable for practical applications with 10 – 100′s of tasks and resources.143,144

In some cases, one can stop the brute-force search once a “good enough” solution is found and thus bound the cost of the algorithm, while not getting an optimal solution. There are also approaches that use dynamic programming (DP), wherein the original scheduling problem is decomposed into a set of overlapping subproblems, each of which can be solved optimally in tractable time. Then, the solutions to the subproblems are directly reused using “memoization” when exploring the search space. For example, RTBA uses DP to construct a strategy table for a task of a given size and deadline.
constraint, which is used to schedule all tasks that fall within this size and deadline. This has a time complexity of $\mathcal{O}(C^2 T_d)$, where $T_d$ is the deadline and $C$ is the compute time, and this is faster than the brute-force approach that has a complexity of $\mathcal{O}(A|T_d|^2)$, where $A$ is the set of possible scheduling actions. As we see, DP gives optimal solutions much faster than brute force. However, it is still unlikely to be fast enough for practical use in online scheduling.

Divide and conquer algorithms partition the scheduling problem into smaller nonoverlapping subproblems and solve these subproblems recursively. The solutions to the subproblems (tasks) are combined into a solution to the original problem. In practice, such optimal solutions are used for a small problem that then contributes to an overall approximate solution. Deng et al\cite{Deng2018} decomposed the overall problem of distributing a workload among fog and cloud resources into subproblems using an approximate and solve these subproblems individually using different optimization techniques. Alternatively, the optimal solution, while impractical, offers a theoretical baseline against which to empirically compare their proposed approximate solution.\cite{Ghosh2014,Simhnan2017}

Consequently, scheduling problems are often solved using suboptimal algorithms that resort to heuristics or metaheuristics that, in practice, come close to optimal solutions but within a reasonable time. Greedy algorithms select the most promising option from the solution space at any given stage. It makes a series of locally optimal choices with the expectation of finding the global optimum. Greedy DAG scheduling heuristics such as heterogeneous earliest-finish-time (HEFT) schedule DAGs over grid resources,\cite{Varshney2009} and have been reused for scheduling on heterogeneous cloud VMS to give expectation of finding the global optimum. Greedy algorithms select the most promising option from the solution space at any given stage. It makes a series of locally optimal choices with the expectation of finding the global optimum. Greedy DAG scheduling heuristics such as heterogeneous earliest-finish-time (HEFT) schedule DAGs over grid resources,\cite{Varshney2009} and have been reused for scheduling on heterogeneous cloud VMS to give short makespan.\cite{Deng2018,Varshney2009} HEFT has a complexity of $\mathcal{O}(E.R)$ where $E$ is the number of edges in the DAG and $R$ is the number of resources. These have also been extended to include cost/time budgets/goals using functions like GAIN and LOSS.\cite{Ghosh2014,Simhnan2017,Zamboni2018}

There are greedy algorithms for scheduling BoTs on cloud resources\cite{Bittencourt2018} and DAG scheduling on edge devices.\cite{Bittencourt2018,Varshney2009} They use heuristics such as predicting the proximity between mobile edge devices for deadline planning, prioritizing tasks with the most resource constraints, and incrementally colocating tasks on the same resource to avoid network latency. More sophisticated heuristics have also been proposed, such as the ToF planner and AutoBoT that uses a greedy approach to minimize the monetary cost for executing a bag of workflows or tasks on clouds.\cite{Ghosh2014,Simhnan2017} Similarly, backtracking algorithms follow one of many possible alternatives and backtrack if it does not look promising.\cite{Deng2018,Varshney2009}

Workflow scheduling problems are often reduced to integer linear programming (ILP) and its mixed variant.\cite{CloneCloud2018} For example, CloneCloud\cite{CloneCloud2018} uses ILP to find partition points between edge and cloud in the application that minimize the overall execution time or energy consumption. While standard techniques exist to provide optimal solutions to ILP problems, they can only be used for small-sized problems due to their prohibitive computational cost. So, heuristics are used to solve these ILP problems as well.\cite{CloneCloud2018,Simhnan2017,Varshney2009}

Metaheuristics are a class of high-level guidelines that can be used to define heuristics to solve a wide class of optimization problems.\cite{Simhnan2017} Specific scheduling problems are refactored to fit the higher-order problem, after which the heuristic guidelines can be applied to explore the solution space. Metaheuristics have been categorized into trajectory-based and population-based methods, with simulated annealing falling in the former category, and genetic algorithm (GA), particle swarm optimization (PSO), and ant colony optimization (ACO) featuring in the latter method. For example, in the work done by Rodriguez and Buyya,\cite{Rodriguez2018} a deadline-constrained workflow scheduling on clouds is mapped to a PSO problem. A particle’s coordinates encode the mapping between the task and a resource, and the dimension of the particle matches the number of tasks. The fitness function for the PSO, which has to be minimized, is the total execution cost, and the schedule is generated by solving this PSO problem with a complexity of $\mathcal{O}(P.M^2.R)$ per iteration, where $P$ is the number of particles, $M$ is the number of tasks, and $R$ is the number of resources. In the work done by Ghosh and Simmhan,\cite{Ghosh2014} a DAG scheduling problem across edge and cloud has been reduced to GA formulation. Each chromosome represents a mapping function from a task to an available edge or cloud resource. The chromosome that gives the minimum makespan, among all the solutions that do not violate compute and energy constraints, is selected from across generations.

Scheduling heuristics are also tuned for operating across resource abstraction layers, such as tasks across mobile edge and cloud layers\cite{CloneCloud2018} or DAGs across fog and cloud.\cite{Bittencourt2018} Some also use locality of tasks to layers. Bittencourt et al\cite{Bittencourt2018} scheduled DAGs that are submitted to a cloudlet, and the scheduler can execute it on the “local” or a remote fog, and/or cloud resources. Likewise, Ghosh and Simmhan\cite{Ghosh2014} scheduled transactional DAGs on edge and cloud resources but pinned the source tasks to the edge and the sink tasks to the cloud.

### 3.4.2 Type of schedule

The scheduling algorithm can be designed as a static (offline) algorithm that is run once when the scheduling unit arrives or a dynamic (online) algorithm that actively decides the schedule based on the current conditions through the lifetime of the unit.
- **Static schedule** In a static approach, the mapping from the scheduling unit to resource(s) is generated once before the unit starts, based on the information about other units and the resources available to the scheduling algorithm a priori. This allocation is retained for the lifetime of the task or DAG and does not respond to changes in the resources or tasks at runtime. It assumes that the prior knowledge is perfect. It also does not make use of checkpointing and resubmission.

Abrishami et al.\textsuperscript{56} generated an offline schedule for a deadline-constrained DAG. It assigns a partial deadline to tasks in the critical path of the DAG, along with their earliest start time, earliest finish time, and latest finish time (LFT).\textsuperscript{59} The algorithm then recursively assigns all the tasks on the path to a single VM, which can finish each task before its LFT, with the minimum price. SPSS also uses static scheduling to schedule an ensemble of workflows.\textsuperscript{87} They use admission control to prevent scheduling of workflows which cannot complete within the deadline and budget constraints. While static algorithms do not consider runtime dynamism, Calheiros and Buyya\textsuperscript{63} mitigated the effects of variations in performance using task replication. It uses the budget surplus and the idle slots in the allocated resources, after performing a static schedule, for task replication. This increases the chances of the deadline being met.

Static scheduling is also possible for a transactional mode of submission. Many cloud providers including Amazon, Google, and Azure offer users rule-based autoscaling for transactional tasks. Here, incoming tasks are routed, typically, in a round-robin manner to a pool of active VMs based on user rules that decide when and how to increase or decrease the number of VMs in this pool based on their utilization. These can efficiently handle the dynamic task-based transactional workloads if the user is able to select the right threshold value.\textsuperscript{62,151} Similarly, sensor and event-based applications have a dynamic workload pattern that schedulers adapt to on edge and fog resources.\textsuperscript{12,17} However, some autoscaling mechanisms use future workload prediction by monitoring current resource utilization to perform proactive autoscaling.\textsuperscript{152}

Static schedules are useful when the workload is known in advance and the performance of resources is deterministic. Real-time applications, those with a closed control loop and those sensitive to mobility, require careful design. These may prefer a static schedule to ensure determinism, preclude the use of mobile resources, and/or retain the decision logic in a single resource.\textsuperscript{153} However, static planning, exclusively, cannot respond to faults and dynamism in compute and network performance, acquisition time, and spot prices on edge, fog, and cloud resources. These can cause suboptimal QoS performance or, worse, violation of hard constraints.

- **Dynamic schedule** Dynamic algorithms use runtime knowledge about the resource performance and application behavior, besides static knowledge, to make scheduling decisions at the start and during the lifetime of the scheduling unit. This information helps them adapt their schedule on the fly to avoid QoS constraint violations, and/or to improve the optimization goals. Dynamic algorithms can either be just-in-time, where the scheduling decision for a task in a workflow is made just once before the task’s execution (but after the DAG itself has started executing), or fully dynamic, where running tasks can be remapped from one resource to another during its execution.

DPDS maintains a priority queue of ready tasks from a batch of DAGs, ordered by the workflow priority, that are scheduled just-in-time.\textsuperscript{87} Whenever a VM is idle, the ready task at the head of the queue is assigned to it. Once a task completes, other dependent tasks in the DAG which are ready are added to this queue and the scheduling continues. A similar strategy is used for scheduling multiple workflows that are submitted in a transactional mode.\textsuperscript{128} The tasks in the queue are mapped just-in-time to the best VM to ensure subdeadline and subbudget are not violated.

Other just-in-time heuristics schedule workflow tasks onto both spot and on-demand VMs.\textsuperscript{89} At runtime, it decides for each ready task whether to schedule it on a spot VM or an on-demand VM based on its latest time to on-demand (LTO), i.e., the slack time before which they must be executed, to avoid exceeding the deadline. Tasks that are ready before their LTO are scheduled on spot VMs while ones that arrive after their LTO are mapped to on-demand VMs. Elsewhere, the progress of each task is continuously monitored, and if one gets delayed, it is dynamically rearranged and future tasks rescheduled to prevent an increase in the overall makespan of the application.\textsuperscript{117}

RTBA uses a fully dynamic algorithm, which actively manages the life cycle of individual tasks on spot and on-demand VMs.\textsuperscript{88} It statically constructs a strategy table, as discussed earlier, and performs dynamic lookups on this table to decide actions based on the current spot price and task progress. The actions can checkpoint and migrate the task, change the bid price, or even pause the task for the bid price to drop. Similarly, CloneCloud\textsuperscript{99} initially builds a partition database for the application, and a partition configuration is selected from this at runtime based on the current resource availability and network conditions. The ToF planner combines both static and dynamic strategies to schedule transactional workflows.\textsuperscript{101} It accumulates workflows arriving for a certain time period and statically assigns a provisional set of VMs to their tasks. However, the VM instance itself is only decided (or started) at runtime to enable the reuse of running VMs.

Applications that are scheduled on edge and fog resources need to be particularly responsive to resource dynamism that is more acute on these platforms.\textsuperscript{15,22} This also requires the runtime capabilities for monitoring to determine when
such adaptation is required.\textsuperscript{19} This may even require changing the coordination strategy from, for example, centralized to a federated or P2P one.

### 3.4.3 Algorithm evaluation

Finally, we consider how the proposed scheduling algorithm is evaluated for its ability to meet the QoS for the application and system models. Others\textsuperscript{33} already identify some of these categories of evaluations, such as experimental, analytical, simulation, and combinations of these. These used in our classification for completeness.

Some papers offer both analytical proofs and experimental validation to measure the efficiency of their scheduling algorithm.\textsuperscript{136} Several others consider only an empirical evaluation using real-world scientific applications such as Cybershake, Genome, LIGO, and Montage,\textsuperscript{86,87,89,101} or domain-specific workloads.\textsuperscript{154} For scheduling across edge, fog, and cloud, mobile applications such as virus scanning, image search, and video analytics have been used for empirical evaluation.\textsuperscript{15,50,155} However, the predominant choice of validation in literature is limited to simulations based on synthetically generated BoTs and DAGs.\textsuperscript{12,146,147}

During simulation, various probability distributions can be used to generate synthetic BoTs and to model the arrival rates of tasks in case of transactional mode.\textsuperscript{12,15,126,147} Random DAG generators are also used to generate DAGs with parameters and distributions to control the number of tasks, number of dependency edges, range of task length, depth and width of DAG, outdegree, and communication to computation ratio.\textsuperscript{117,122,156} Alternatively, publicly available cloud workloads, such as the Google Cluster Workload also offer the choice of realistic DAG and BoT simulations.\textsuperscript{121,157} The CloudSim\textsuperscript{158} and iFogSim\textsuperscript{159} simulation frameworks can model application execution on cloud infrastructure and edge and fog infrastructure, respectively, and are often used to evaluate scheduling algorithms.\textsuperscript{23,24,38,87,89} Many algorithms consider system models that are realistic and similar to those provided by major cloud providers. AWS is a popular choice here, with many considering different EC2 on-demand VM types and their prices and configurations in their simulation.\textsuperscript{62,63,110} Some\textsuperscript{12,15} also use real-world benchmarks and distributions of network and compute performance of edge and cloud resources.

Algorithms that use preemptible VMs simulate the fluctuation in spot prices using historic spot price data provided by Amazon,\textsuperscript{89,121} and to train their price bidding models and expected lifetime estimation models.\textsuperscript{88,103} The acquisition lag for VMs may be based on a distribution of past startup times\textsuperscript{116,160} or limited to a constant value.\textsuperscript{62,87} Similarly, the mobility of edge and fog devices can be captured using real-world mobility traces or generated synthetically using mobility models. Previous studies\textsuperscript{15,17} simulated random vehicle motion for a vehicle-to-vehicle streaming application, where each vehicle randomly picks another vehicle to start streaming a video to it.

### 4 CLASSIFICATION OF SCHEDULING ALGORITHMS USING TAXONOMY

Table 1 classifies 36 key publications on application scheduling, based on the taxonomy we have proposed. Twenty-five of these papers propose scheduling exclusively on cloud resources, while 11 use a mix of edge, fog, and/or cloud. The skew toward cloud-based scheduling reflects the decade-long period of maturity of this resource, relative to edge and fog resources that are more recent. The footnotes of the table indicate the shorthand of the taxonomy terms used within the columns for brevity.

In summary, on the system model dimension, 23 of the 25 papers consider cloud infrastructure with heterogeneous VM sizes, 17 use on-demand VMs with fixed prices, 2 use preemptive VMs, and 5 use a mix of both. For the mix of edge, fog and cloud, almost all the papers have considered heterogeneous resources and only one paper considers homogeneous cloud resources.

A majority of the cloud scheduling publications, 19 to be exact, use hourly billing that was common among cloud providers, while two others use more current fine-grained billing intervals. Further, 12 of the papers cataloged consider VM acquisition lag in their schedule planning. Three of these papers also constrain the availability of resources from the provider, which is more representative of Grid and HPC systems than public clouds which provide seemingly unlimited resources on-demand. For edge, fog, and cloud resources, most of the papers do not consider resource pricing at all. Only two papers consider hourly billing and that too only for the cloud resources. Also, one paper has considered deployment lag and one has considered deferred acquisition. All these papers constrain the availability of edge and fog resources, either on account of mobility or limited resources capacity, while two limit the number of available cloud resources as well. Three papers consider spatial mobility of edge and fog resources, while two others are have spatial locality between the client and the edge or fog.
| Paper | Layer | System Design Section 3.1 | Application Model Section 3.2 | Quality of Service Section 3.3 | Algorithm Section 3.4 |
|-------|-------|--------------------------|-------------------------------|-------------------------------|-----------------------|
| 56    | C     | HT OD BI-60, 5 min, Linear PPR | DAG S Data transfer time | TD/H MIN(C) DAG, DAG | Divide and Conquer Static S – Synthetic direct acyclic graphs (DAGs) of CyberShake, Epigenomics, LIGO, Montage, SIPHT |
| 107   | C     | HT OFF, OD BI-60 min | DAG S Data transfer time | TD/S MIN(C) DAG, DAG | Greedy Static S – Synthetic DAGs of AIRSN, Chimera, CSTEM, LIGO, Montage, random; E – Image processing application |
| 63    | C     | HT OD BI-60 min, AL, PV | DAG S Resubmit, Replicate, Data transfer time | TD/S or – MAX(O) or MIN(T) DAG, Batch | Backtracking Dynamic S – Random DAGs |
| 117   | C     | HT – Limited availability | DAG [M, HO, T/S] Non-deterministic runtime, Data transfer time | TD/S or – MIN(T) –, DAG | Backtracking Dynamic S – Random DAGs |
| 103   | C     | HO OD, SP BI-60 min, NP(asym.) | DAG S – | – | MIN(T) –, DAG | Heuristic Static E – WordCount, Pi, Sort on EC2 |
| 88    | C     | OD:HT, SP: OFF:HO BI-60 min, NP (asym.), AL | Task S Checkpoint, Migrate, Data transfer time | TD/H MIN(C) Task, Task | Dynamic Programming Dynamic S – Varying length tasks |
| 136   | C     | OD BI (novel pricing model), PV | DAG S Data transfer time | – | MIN(C) and MIN(T) DAG, DAG | Greedy Dynamic H – Synthetic WIEN2k DAG, Random DAGs |

*E: Edge, F: Fog, C: Cloud; M: Spatial mobility.*
*HO: Homogeneous machine sizes; HT: Heterogeneous sizes.*
*OD: On-demand fixed price virtual machines (VMs); SP: Preemptible spot/variable price VMs; OFF: Off-cloud resources; FR: Free edge/fog resources.*
*B: Billing interval; AL: Acquisition lag considered; PV: Performance variations considered; NP: Network pricing; PPR: Different price to performance ratios considered.*
*S: Single scheduling unit submitted; M: Multiple units; HO: Homogeneous units; HT: Heterogeneous units; B: Batch submission; T: Transactional submission.*
*(S): Single granularity of scheduling; /O: Ordered queue; /U: Unordered bag.*
*T: Time deadline, CB: Cost budget; E: Energy constraint; S: Spatial constraint; /H: Hard constraint; /S: Soft constraint.*
*MIN: Minimize objective function; MAX: Maximize function; (T): Time; (C): Cost; (O): Other.*
*Gran.: Granularity of QoS constraint and granularity of quality-of-service (QoS) goal.*
*A: Analytical; E: Empirical; S: Simulation; H: Hybrid.* (Continues)
| Paper | System Design Section 3.1 | Application Model Section 3.2 | Quality of Service Section 3.3 | Algorithm Section 3.4 |
|-------|--------------------------|-------------------------------|-----------------------------|----------------------|
|       | Layer a | Size b | Price c | Char. d | Mode e | Char. f | Constr. g | Goal h | Grnn. i | Technique | Schedule | Evaluation j |
| 147   | C       | HT     | OD      | BI-60 min | Task                              | $\{M, HO, BJ/O\}; \{M, HO, T\}/S$ | –             | MIN(T) and MIN(C) | –, Batch (overall BoTs) | Greedy | Dynamic | $S – Random BoTs$ |
| 133   | C       | HT     | OFF, OD | BI-60 min, AL, PV, Failures | Task                              | $\{M, HO, T\}/O$ | Resubmit, Non-deterministic runtime | TD/S or CB/S | MIN(T) or MAX(O) | Batch, Batch | Greedy | Dynamic | $E – DAG of Reservoir simulation ensemble, Kalman filter on TeraGrid, EC2$ |
| 104   | C       | HT     | SP      | BI-60 min | Task                              | S | –             | –             | MIN(C) | –, DAG | Heuristic | Static | $S – Varying length tasks$ |
| 146   | C       | HT     | OD      | –        | DAG                              | S | Data transfer time | –             | MIN(T) | –, DAG | Greedy | Static | $S – Random DAGs$ |
| 138   | C       | HT     | OD      | BI-N/A, Linear PPR, Failures, Limited availability | DAG                              | $\{M, HO, BJ/O\}$ | Resubmit, Data transfer time | TD/S | MIN(C) | Batch, Batch | Divide and Conquer | Dynamic | $S – Synthetic DAGs based on practical workflows such as bank cheque workflow processing$ |
| 87    | C       | HO     | OD      | BI-60 min, AL | DAG                              | $\{M, HO, BJ/O\}$ | Non-deterministic runtime | TD/H and CB/H | MAX(O) | Batch, Batch | Heuristic (DPDS, WA-DPDS), Divide and Conquer (SPSS) | Dynamic (DPDS, WA-DPDS), Static (SPSS) | $S – Synthetic DAGs of CyberShake, Epigenomics, LIGO, Montage, SIPHT$ |
| 62    | C       | HT     | OD      | BI-60 min, AL | DAG                              | $\{M, HO, T\}/S$ | Non-deterministic runtime | TD/S | MIN(C) | DAG, Batch | Divide and Conquer | Dynamic | $S – Pipeline, Parallel, Hybrid DAGs$ |
| 127   | C       | HT     | OD      | BI-60 min, AL | Task                              | $\{M, HO, BJ/U\}$ | Resubmit, Non-deterministic runtime | CB/S | MIN(T) | Batch, Batch | Dynamic Programming | Dynamic | $S – BoT w/ Normal Distribution$ |
| 89    | C       | OD:HT, SP-HO | O, SP     | BI-60 min, AL, PV | DAG                              | S | Checkpoint, Data transfer time | TD/S | MIN(C) | DAG, DAG | Heuristic | Dynamic | $S – Synthetic DAG of LIGO$ |
| 110   | C       | HT     | OD      | BI-60 min, SP | DAG                              | S | Resubmit, Replicate, Data transfer time | TD/S | MIN(C) | DAG, DAG | Heuristic | Dynamic | $S – Synthetic DAG of LIGO$ |

Tasks of a BoT are submitted in a batch and scheduled as an ordered collection.

Multiple BoTs are submitted transactionally and each BoT is scheduled as a single unit as soon as it arrives.

(Continues)
| Paper | System Design Section 3.1 | Application Model Section 3.2 | Quality of Service Section 3.3 | Algorithm Section 3.4 |
|-------|--------------------------|-----------------------------|-------------------------------|----------------------|
|       | Layer^a | Size| Price| Char.| Struct.| Mode| Char.| Constr.| Goal| Gran.| Technique| Schedule| Evaluation^i |
| 116   | C | HT | OD | Bi-60 min, AL, PV | DAG | S | Non-deterministic, runtime Data transfer time | TD/S | MIN(C) | DAG, DAG | PSO | Static | S – DAGs of CyberShake, LIGO, Montage, SIPHT |
| 106   | C | HT | OD, SP | Bi-60 min, AL | Task | S | Resubmit, Replicate, Checkpoint, Migrate | – | MIN(C) and MAX(O) | –, Task Heuristic | Dynamic | S – Random tasks, Google cluster trace on EC2 |
| 113   | C | HT | OD | Bi-60 min, different PPR, AL | Task | {M, HO, B}/U | – | TD/H or CB/H | MIN(C) or MIN(T) | Batch, Batch | Heuristic | Static | S – Random DAG |
| 105   | C | HT | SP | Bi-60 min | Task | {M, HO, T}/O | Replicate, Checkpoint, Migrate, Data transfer time | TD/S | MAX(O) and MIN(C) | Task, Batch | Heuristic | Dynamic | S – Synthetic tasks based on job stream from LHC Grid |
| 122   | C | HT | OD | Bi-per unit time, PV w/ bounded randomness, PPR | DAG, Task | Service | {M, HO, T}/S; Task {M, HT, T}/U | TD/H and TB/H | MIN(C) or MIN(T) or (MIN(C) and MIN(T)) | DAG, Batch | Service-Heuristic; Task–GA, PSO, ACO | Static+ dynamic | S – Random DAGs |
| 128   | C | HT | OD | Limited availability | DAG | {M, HO, T}/O | – | TD/S and CB/S | MIN(C) or MIN(T) and MAX(O) | DAG, Batch | Divide and Conquer | Dynamic | S – Random DAGs |
| 156   | C | HT | OD | Bi-60 min, AL | DAG | S | Data transfer time | CB/H | MIN(T) | DAG, DAG | Greedy | Static | S – Random DAGs |
| 101   | C | HT | OD | Bi-60 min | DAG | {M, HO, T}/O | Checkpoint | TD/S or CB/S | MIN(C) or MIN(T) | DAG, Batch | Greedy | Static+ | S – Synthetic DAGs of LIGO, Montage on EC2, Rackspace |

(Continues)
| Paper | System Design Section 3.1 | Application Model Section 3.2 | Quality of Service Section 3.3 | Algorithm Section 3.4 |
|-------|--------------------------|-----------------------------|-------------------------------|-----------------------|
| Layer | Size | Price | Char. | Struct. | Mode | Char. | Constr. | Goal | Gran. | Technique | Schedule | Evaluation |
| 15 E/M | HT | - | E - Limited availability | DAG | [M, HO, T/O] | Resubmit, Migrate, Data transfer time | TD/H | MIN(E) or MIN(T) | DAG, MIN(E)-Batch; MIN(T)-DAG | Greedy | Dynamic | S, E – Face detection and speech to text applications |
| 17 E/M, F, C | HT | - | E - Limited availability | DAG | S | Migrate | S/H | MIN(T) | DAG, DAG | Heuristic | Dynamic | S – Vehicle-to-vehicle video streaming and mobile CEP applications |
| 66 F, C | HT | - | F - Limited availability | DAG | S | - | S/H | - | DAG, - | Heuristic, Backtracking, Greedy | Static | S – Fire alarm application |
| 161 E/M, C | HT | NP | E-VM migrate, Time slots, Locality | Task | [S, HT, T/O] | Locality, Tasks on specific Edges | TD | MIN(O): NW use | Task | Markov Decision Process/ILP | Static | S – Real mobility traces, cellular NW |
| 38 F, C | HO | - | F - Limited availability, locality | DAG | [S, HT, T] | DAG locality, Data transfer time | - | MIN(T), MIN(O): NW use | DAG | Locality-aware, FIFO, Heuristic | Dynamic | S – iFogSim, gaming video processing |
The reviewed papers have diverse application characteristics as well. Out of all the papers, 25 use a DAG structure, while 12 use a task structure; seven publications allow batch submission while 14 have a transactional mode of arrival; and 15 consider applications that are scheduled as a collection rather than individually. As many as 13 of the papers use time deadline as a constraint and cost as the optimization goal when specifying their QoS requirements. However, eight articles use cost budget as a constraint as well. Most of the papers that include edge and fog resources aim at minimizing the power consumption, overall latency, or network usage.

Most of the research papers use a heuristic or greedy algorithm for scheduling, of which 16 schedule the applications statically, 19 schedule them dynamically, and 2 use a mix of both approaches. Simulations based on synthetic DAGs or tasks tend to be the predominant means of evaluating these scheduling algorithms, although five of the papers reviewed use real or realistic applications for validation.

This diversity in the surveyed literature indicates the rich classes of problem definitions and corresponding research outcomes in this area of application scheduling on edge, fog, and cloud. It also justifies the need for a detailed taxonomy such as ours to categorize and analyze this body of work, ie, both what has been done and future work that can benefit from these learnings, particularly for a mix of these resource abstractions.

5 | RELATED WORK

Several existing works offer generic reviews of edge, fog, or clouds, either independently or in comparison. They also offer complementary aspects of application models on each resource independently, or scheduling for other levels of the cloud abstractions. However, none offer a holistic survey on the approaches to application scheduling on these distributed infrastructure, with structure and rigor as we have presented here.

5.1 | Resource characterization

There are several surveys on the characteristics of edge, fog, and cloud resources. These describe the capabilities of the individual resource platforms and their features that benefit both end users and service providers. This is necessary to examine application scheduling but needs to be substantiated with applications models and scheduling techniques that leverage these feature.

A recent manifesto on cloud computing offers an overarching review of contemporary cloud capabilities and future potential and challenges. Our cloud resource characteristics are not as detailed but focus on specific features of relevance to application scheduling. Coutinho et al offered a more specialized bibliometric survey on elasticity of cloud resources, along with the journals where the publications appear, country of origin of authors, and year of publication. A taxonomy of methods used to leverage as well as QoS metrics is also provided. While some of our resource characteristics, application characteristics, and QoS topics overlap with this, we do not exclusively focus on elasticity but rather examine other dimensions of clouds such as pricing and resilience as well, in addition to application scheduling.

Similarly, there have been several papers that conceptualize the idea of fog computing and cloudlets. Others examine specific applications that benefit from the fog layer, to complement edge and cloud computing, with smart cities and IoT being key drivers. Some also prescribe different types of interactions between the edge, fog, and cloud layers. These are similar to our resource taxonomy but fail to compare common and contrasting features across edge, fog, and cloud, and how application models and schedulers leverage them.

There are articles that discuss the use of edge, fog, and cloud layers in a hierarchical architecture. Masip-Bruin et al considered mobility to be a characteristic feature of both the edge and fog layers, with challenges to resource discovery, service scheduling, QoS guarantee, and security. A medical emergency use case is used to illustrate the relative benefits of using a cloud-only application deployment design, with one that uses all three. Latency is seen as a QoS goal, but they omit concerns on monetary cost and energy usage. Similarly, Yu et al highlighted the security and privacy issues in including edge resources for storage and computation of IoT applications, besides the cloud. A fog layer is not explicitly considered, and resource pricing is mentioned in passing. The benefits for different IoT applications such as smart grid, smart city, and smart transportation are mentioned. Our survey goes beyond exemplars and considers how generic application can be conceptually specified in terms of their structure, mode of submission, and QoS. We also provide a categorization of possible scheduling algorithms in literature. Individual and combined resource layers are included in our review, without limiting to a hierarchical model.
MEC has received particular attention due to early mobile phones that were resource constrained. Mac and Becvar\textsuperscript{27} surveyed the existing research on computation off-loading in MEC and how it is integrated with the mobile network architectures. They illustrated use cases that benefit from MEC and their application characteristics, such as whether the application can be partitioned and offloaded, dependencies between parts, and predictability of the input data size. These affect how an application can be executed locally on the phone, partially offloaded or fully offloaded to the cloud to meet the QoS goals, such as minimization of delay or energy consumption. Techniques for responding to the device’s mobility, such as changing transmission power, VM migration, and communication route selection are suggested. Abbas et al\textsuperscript{26} offered a conceptual model of computing that moves between a mobile edge device and the cloud, with the intelligence on application scheduling present between them on the radio access network. This communication-centric view considers fog and edge computing as the same resource layer. They review literature on computation off-loading in MEC, with low latency processing, storage issues, and energy efficiency being QoS goals. Our survey while similar in spirit to these takes a more holistic and forward-looking view of the resource characteristics of edge, fog, and cloud, including pricing and performance, and generalizes the application structure and their QoS.

5.2 Scheduling techniques

Scheduling for different aspects cloud computing have been examined in the past. Zhan et al\textsuperscript{8} presented a taxonomy of scheduling algorithms at various layers, namely, application, virtualization, and infrastructure. Algorithms are classified based on the specific objectives that should be met at each layer. Scheduling at the virtualization layer has the goal of mapping VMs on to physical machines, which is of particular interest to service providers. This has been addressed by other specialized surveys.\textsuperscript{29,163} The goal of scheduling at the infrastructure layer is to place the resources and services at different locations in various data centers. Grozev and Buyya\textsuperscript{10} drilled down into this layer and offered a taxonomy for intercloud architectures, with application brokering in these systems. At the application layer, the problem is to schedule the user’s application on VMs, and three goals are discussed by Zhan et al\textsuperscript{8}: user QoS, provider efficiency, and negotiation.

In contrast, our survey focuses on just the application layer but goes in depth by offering a detailed classification of edge, fog, and cloud resources based on pricing models and system characteristics essential for scheduling algorithms. Moreover, we also categorize the structure of workflows, their characteristics, and mode of submission. We further identify fault tolerance as a user QoS with scheduling techniques designed for application resilience on these resource layers. Our work is of broader interest to middleware and application developers using these distributed resources, who, arguably form a larger population that can put this survey to practical use, compared with commercial resource service providers who are less influenced by such research outcomes.

Similarly, Kessaci et al\textsuperscript{16} also discussed scheduling at the three levels, namely, service, task, and VM, for public and private clouds. We limit our focus to public clouds that are exceedingly popular and where issues of elasticity and costing offer clear challenges and opportunities to end users, and complement this with emerging edge and fog resource layers. Our work also emphasizes the application layer, a subset of which is the task level considered by Kessaci et al.,\textsuperscript{16} with the objective of meeting QoS requirements and/or budget constraints for the application. We also offer greater detail on various aspects of the unit of scheduling, which is lacking in these surveys. Likewise, the work of Huang et al\textsuperscript{34} limits itself to resource allocation in clouds in the presence of system failures, with various ACO- and GA-based dynamic scheduling metaheuristics to handle faults being explored. We go beyond system failure and examine literature that handle failures due to pricing (out of bid) for preemptible and opportunistic resources. We also discuss several other scheduling algorithms that address diverse QoS constraints and goals.

Wu et al\textsuperscript{11} presented several categories of workflow scheduling algorithms on clouds, including static and dynamic scheduling, to meet constraints such as budget, deadline, and robustness. Similarly, Liu et al\textsuperscript{141} offered a brief summary of workflows and their objective criteria for scheduling on the cloud and reviewed literature on scheduling algorithms to achieve the same. Others have surveyed metaheuristic techniques for scheduling workflows on the cloud.\textsuperscript{32} These techniques include hill climbing, simulated annealing, tabu search, GA, PSO, and ACO, and these have been used to generate a schedule for workflows on clouds. While these surveys consider multiple classes of scheduling algorithm and system characteristics that lie at the intersection of several dimensions we introduce, they do not offer a taxonomy of the dimensions themselves, which is essential for a rigorous analysis of this space. Edge and fog resources, which have lately gained prominence, are omitted as well.

Some early research investigates platform and application models for edge and fog computing. Hong et al\textsuperscript{17} and Varshney and Simmhan\textsuperscript{22} proposed a three-level strictly hierarchical model where the computation is rooted in the
cloud, resources are elastically acquired in the cloud and fog layers, and communication is possible between cloud and fog, or fog and edge. However, their example applications do not use the cloud, and this degenerates to a client-server model between the edges and their fog parent. The role of virtualization in enabling cloud computing is discussed by Bit tencourt et al\textsuperscript{72} and they see a similar role for the fog as well. They conceived of a VM encapsulating all dependencies for an edge application or user to be hosted on a Cloudlet within one hop of the edge, with this VM migrating to remain at one-hop distance from the edge user. Such articles offer potential architectures for interactions between edge, fog, and cloud, while our survey more broadly characterizes these resources and examines their impact on applications and how they are scheduled.

6 \ DISCUSSION

This detailed review of application scheduling characteristics on edge, fog, and cloud resources, along with the feature matrix, highlight several open problems, whose solutions require a mix of research, development, and business models. There are also rapidly emerging technologies that can influence these directions. We discuss these along similar categories that we have proposed.

6.1 \ Evolution in resource abstractions

Public cloud providers are highly agile and respond rapidly to evolving technologies and market dynamics. One challenge is being addressed is lightweight application sandboxing, in contrast to hypervisor-based VMs. Cloud providers such as Amazon EC2 and Microsoft Azure are offering Docker containers, which have minimal resource overheads and offer rapid instantiation compared with virtualization and are useful when OS heterogeneity is not required. They even support basic migration between hosts. These containers however use kernel-based controls to enforce security and resource allocation, which are less effective than hardware virtualization. This can open up opportunities for trade-off between the ability to respond rapidly to application dynamism, multitenant container security, and handling performance variability.

Recent work on minimalist OS such as VMWare’s PhotonOS, lightweight virtualization such as AWS Firecracker, and even web standards such as W3C’s WebAssembly, offer alternatives to containers and hypervisors. Some of these are motivated by the popularity of a serverless FaaS model, based on stateless microservices that encapsulate user-defined functions that can be composed and executed.\textsuperscript{164} This is similar to task scheduling in a transactional model. Besides cloud data centers, FaaS is also being pushed to the fog and edge layers using SDK’s like Amazon Greengrass and Azure Edge IoT offered by the cloud providers. These cloud fabrics extend to edge devices and allow for more centralized management of distributed edge and fog resources on the wide-area network.\textsuperscript{162,165,166} These are currently limited to running applications using a FaaS model, with their scheduling managed exclusively by the provider. However, the ability to expose IaaS resources on the edge and fog can help further leverage scheduling designed for the cloud to be extended to the edge and fog, and ease the design of practical edge, fog, and cloud applications.

In addition to the push of existing cloud providers, there are also alternative business and technology models for edge and fog computing that can be sustainable.\textsuperscript{68} both for infrastructure deployments and platform support.\textsuperscript{22} They are more obvious in vertically integrated “private” scenarios rather than horizontal, reusable “public” ones, much in the way of cloud data centers evolving from private use by Amazon, Google, and Microsoft to a commercial business model of a public cloud.\textsuperscript{40,162} Potential providers of on-demand public fog computing are operators of cell phone towers who have captive power, communications, and space, and smart city deployments with captive compute capacity as part of the city deployments of verticals such as smart power grid or smart transportation.\textsuperscript{36,39,167}

Likewise, the advent of energy-efficient and high-performance accelerators, such as GPUs and tensor processing units, and low-power ARM64 servers as cloud and fog resources introduces further resource diversity and application opportunities that impacts scheduling.\textsuperscript{162,168,169} Part of this is driven by the rapid adoption of machine learning models and deep neural networks, which analyze multimedia data (eg, video surveillance from smart cities) and have high computing costs.\textsuperscript{170} Novel edge and fog devices such as drones and other autonomous vehicles are starting to become a reality as well,\textsuperscript{171} and introduce new challenges in the energy and compute-constrained mobile resources with transient communications. Likewise, the adoption of 5G communication technology can translate into widespread deployment and accessibility of edge and fog devices, offering a high-bandwidth and pervasive last-mile link.\textsuperscript{28} These technological shifts will require us to revisit many of the assumptions on the system models used for scheduling.
At the same time, there is also a lack of standardized infrastructure and platform interfaces for edge and fog computing, with much of the advances in fabric management, programming models, power and network management, fault tolerance, and pricing models being limited to research prototypes. However, initiatives such as OpenFog Consortium (which recently merged with the Industrial Internet Consortium) and EdgeX Foundry championed by various industries are starting to offer reference architectures and software stacks to address this gap. These will serve as the vehicle to incorporate and enact the scheduling models that are developed.

6.2 Application Models and QoS

Application models tend to evolve more slowly than hardware and communications technologies, which are driven by the industry. Research has tended to focus more on batch execution of workflows as the unit of scheduling. However, the growth in streaming data and online decision-making applications means that transactional workloads and event-driven models need to be better examined. In fact, processing streaming data and having a control loop between sensors and actuators, with analytics scheduled in between, is a common pattern in IoT applications. A few of the literature we have tabulated consider such an event-driven or transactional model. Similarly, BoTs are a common abstraction that are inadequately examined for scheduling, even as they make good candidates for off-loading to the cloud or fog partitions for execution. The popularity of FaaS also pushes data to the compute, rather than the typical model of moving compute to the data, while easing weak scaling of stateless microservices.

The increasing importance of machine learning applications means that scheduling that is sensitive to the goals of training and inferencing will be beneficial. In particular, the QoS goals and constraints may need to include the quality of the training and inferencing accuracy as first-class metrics, besides the time taken. Further, with increasing personally identifiable data being collected and processed from the edge, privacy, and trust start playing a key role. This may impose limitations on where to run what applications and if additional operations such as anonymization or masking is required before placing specific tasks on specific resources. This can require the application to be recomposed on the fly before being scheduled.

Scheduling across different resource layers also introduces independent constraints and priorities on each layer that need to be met. For example, energy is a key concern for scheduling on edge and fog, while pricing and latency are user concerns when using the cloud. Additional research is required into capturing these resource-specific factors into the optimization goals or constraints.

6.3 Scheduling approach

While there has been a lot of conceptual work on making use of edge, fog, and cloud resources, there is a lack of literature on novel scheduling approaches that consider all the three layers together while leveraging their relative merits. Further, the interactions between the layers currently tends to be limited to a hierarchy of cloud–fog–edge, or just a flat logical structure composed of heterogeneous resources across them. This is a need to examine distributed scheduling strategies rather than centralized ones to ensure scaling to thousands of resources on the local- and wide-area network, and also resilience to avoid a single point of failure. Techniques from P2P can play a role here.

Given the challenges in access to large-scale edge and fog deployments, much of the validation of scheduling approaches for these resources are based on simulations, using frameworks such as iFogSim. However, there has been work on virtual environments like VIoLET that use containers running on VMs or clusters to replicate the behavior of edge, fog, and cloud resources using container resource allocations. It can also control the network topology between the resources and the resource dynamism and failures. These have the benefit of allowing real applications to be scheduled and executed, and the schedule evaluated for realistic edge, fog, and cloud resources, and offer a balance between empirical and simulation-based approaches. More such efforts are required to model communication diversity, device reliability, etc.

7 CONCLUSION

In this survey, we have offered a comprehensive taxonomy for defining and designing application scheduling algorithms on edge, fog, and cloud infrastructure, based on a detailed literature review. These span the system model for the three layers, application model with an emphasis on DAG and task-based applications, and the QoS goals and constraints to
be met, which collectively help define scheduling as an optimization problem. We also categorize existing approaches to solving this optimization problem and evaluating it, based on prior reviews. This taxonomy has been used to tabulate the characteristics of 36 research papers on scheduling on edge, fog, and/or cloud resources. The taxonomy presents designers of scheduling algorithms for edge, fog, and cloud applications with a clear set of system and application features they should consider for their target infrastructure and workload. The table provides architects of application runtimes with the relevant classes of scheduling algorithms that they can leverage to meet the needs of their end users. Learning from this body of work is essential as applications are increasingly designed for edge and fog environments, rather than just the cloud, and scheduling challenges are set to emerge within newer application platforms that can operate on these resources. To this end, we have also explored various emerging technology trends that will require us to reexamine current system and application models, and develop novel scheduling techniques for the next generation of computing.

**ORCID**

Yogesh Simmhan [https://orcid.org/0000-0003-4140-7774](https://orcid.org/0000-0003-4140-7774)

**REFERENCES**

1. Pinedo ML. *Scheduling: Theory, Algorithms, and Systems*. 3rd ed. New York, NY: Springer Science+Business Media; 2008.
2. Tucker A, Gupta A. Process control and scheduling issues for multiprogrammed shared-memory multiprocessors. *ACM SIGOPS Oper Syst Rev*. 1989;23(5):159-166.
3. Feitelson DG, Rudolph L, Schwiegelshohn U. Parallel job scheduling — a status report. Paper presented at: International Conference on Job Scheduling Strategies for Parallel Processing; 2005; Cambridge, MA.
4. Murata T. Petri nets: properties, analysis and applications. *Proc IEEE*. 1989;77(4):541-580.
5. Taylor IJ, Deelman E, Gannon DB, Shields M. *Workflows for e-Science: Scientific Workflows for Grids*. London, UK: Springer-Verlag London Limited; 2007.
6. Hovestadt M, Kao O, Keller A, Streit A. Scheduling in HPC resource management systems: queuing vs. planning. Paper presented at: Workshop on Job Scheduling Strategies for Parallel Processing; 2003; Seattle, WA.
7. Yu J, Buyya R, Ramamoharan R. Workflow scheduling algorithms for grid computing. In: Xha F, Abrahm A, eds. *Metaheuristics for Scheduling in Distributed Computing Environments*. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 2008;173-214.
8. Zhan ZH, Liu XF, Gong YJ, Zhang J, Chung HSH, Li Y. Cloud computing resource scheduling and a survey of its evolutionary approaches. *ACM Comput Surv*. 2015;47(4):63:1-63:33.
9. Yu W, Liang F, He X, et al. A survey on the edge computing for the Internet of Things. *IEEE Access*. 2017;6:6900-6919.
10. Anderson DP. BOINC: a system for public-resource computing and storage. Paper presented at: 5th IEEE/ACM International Workshop on Grid Computing (GRID); 2004; Pittsburgh, PA.
11. Armbrust M, Fox A, Griffith R, et al. *Above the Clouds: A Berkeley View of Cloud Computing*. Technical report. Berkeley, CA: University of California at Berkeley; 2009.
12. Ghosh R, Komma SPR, Simmhan Y. Adaptive energy-aware scheduling of dynamic event analytics across edge and cloud resources. Paper presented at: 2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID); 2018; Washington, DC.
13. Simmhan Y. Big data and fog computing. In: *Encyclopedia of Big Data Technologies*. Cham, Switzerland: Springer International Publishing; 2018:1-10.
14. Stojmenovic I. Fog computing: a cloud to the ground support for smart things and machine-to-machine networks. Paper presented at: 2014 Australasian Telecommunication Networks and Applications Conference (ATNAC); 2014; Southbank, Australia.
15. Shi C, Lakafosis V, Ammar MH, Zegura EW. Serendipity: enabling remote computing among intermittently connected mobile devices. In: Proceedings of the 13th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc); 2012; Hilton Head, SC.
16. Kessaci Y, Melab N, Talbi EG. Multi-level and multi-objective survey on cloud scheduling. Paper presented at: 2014 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW); 2014; Phoenix, AZ.
17. Hong K, Lilipeth D, Ramachandran U, Ottenwälder B, Koldehofe B. Mobile fog: a programming model for large-scale applications on the Internet of Things. In: Proceedings of the 2nd ACM SIGCOMM Workshop on Mobile Cloud Computing (MCC); 2013; Hong Kong, China.
18. Bonomi F, Milito R, Zhu J, Addepalli S. Fog computing and its role in the Internet of Things. In: Proceedings of the 1st Edition of the MCC Workshop on Mobile Cloud Computing (MCC); 2012; Helsinki, Finland.
19. Dastjerdi AV, Gupta H, Calheiros RN, Ghosh SK, Buyya R. Fog computing: principles, architectures, and applications. In: *Internet of Things: Principles and Paradigms*. Cambridge, MA: Elsevier; 2016:61-75.
20. Chiang M, Zhang T. Fog and IoT: an overview of research opportunities. *IEEE Internet Things J*. 2016;3(6):854-864.
21. Masip-Bruin X, Marin-Tordera E, Tashakor G, Jukan A, Ren G. Foggy clouds and cloudy fogs: a real need for coordinated management of fog-to-cloud computing systems. *IEEE Wirel Commun*. 2016;23(5):120-128.
22. Varshney P, Simmhan Y. Demystifying fog computing: characterizing architectures, applications and abstractions. Paper presented at: 2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC); 2017; Madrid, Spain.

23. Skarlat O, Schulte S, Borkowski M, Leitner P. Resource provisioning for IoT services in the fog. Paper presented at: 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA); 2016; Macau, China.

24. Skarlat O, Nardelli M, Schulte S, Dustdar S. Towards QoS-aware fog service placement. Paper presented at: 2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC); 2017; Madrid, Spain.

25. Deng R, Lu R, Lai C, Luan TH, Liang H. Optimal workload allocation in fog-cloud computing toward balanced delay and power consumption. IEEE Internet Things J. 2016;3(6):1171-1181.

26. Abbas N, Zhang Y, Taherkordi A, Skeie T. Mobile edge computing: a survey. IEEE Internet Things J. 2018;5(1):450-465.

27. Mach P, Becvar Z. Mobile edge computing: a survey on architecture and computation offloading. IEEE Commun Surv Tutor. 2017;19(3):1628-1656.

28. Taleb T, Samdanis K, Mada B, Flinck H, Dutta S, Sabello D. On multi-access edge computing: a survey of the emerging 5G network edge cloud architecture and orchestration. IEEE Commun Surv Tutor. 2017;19(3):1657-1681.

29. Mann ZA. Allocation of virtual machines in cloud data centers–a survey of problem models and optimization algorithms. ACM Comput Surv. 2015;48(1):11:1-11:34.

30. Grozev N, Buyya R. Inter-cloud architectures and application brokering: taxonomy and survey. Softw: Pract Exper. 2014;44(3):369-390.

31. Wu F, Wu Q, Tan Y. Workflow scheduling in cloud: a survey. J Supercomput. 2015;71(9):3373-3418.

32. Tsai CW, Rodrigues JJPC. Metaheuristic scheduling for cloud: a survey. IEEE Syst J. 2014;8(1):279-291.

33. Coutinho EF, de Carvalho Sousa FR, Rego PAL, Gomes DG, de Souza J. Elasticity in cloud computing: a survey. Ann Telecommun. 2015;70(7-8):289-309.

34. Huang L, Chen H, Hu T. Survey on resource allocation policy and job scheduling algorithms of cloud computing. J Softw. 2013;8(2):480-487.

35. Hu P, Dhelim S, Ning H, Qiu T. Survey on fog computing: architecture, key technologies, applications and open issues. J Netw Comput Appl. 2017;98:27-42.

36. Bonomi F, Milito R, Natarajan P, Zhu J. Fog computing: a platform for Internet of Things and analytics. In: Bessis N, Dobre C, eds. Big Data and Internet of Things: A Roadmap for Smart Environments. Cham, Switzerland: Springer International Publishing; 2014:169-186.

37. Satyanarayanan M, Bahl P, Caceres R, Davies N. The case for VM-based cloudlets in mobile computing. IEEE Pervasive Comput. 2009;8(4):14-23.

38. Bittencourt LF, Diaz-Montes J, Buyya R, Rana OF, Parashar M. Mobility-aware application scheduling in fog computing. IEEE Cloud Comput. 2017;4(2):26-35.

39. Stojmenovic I, Wen S. The fog computing paradigm: scenarios and security issues. Paper presented at: 2014 Federated Conference on Computer Science and Information Systems (FedCSIS); 2014; Warsaw, Poland.

40. Yi S, Li C, Li Q. A survey of fog computing: concepts, applications and issues. Paper presented at: 2015 Workshop on Mobile Big Data (Mobidata); 2015; Hangzhou, China.

41. Jin J, Gubbi J, Luo T, Panalaswami M. Network architecture and QoS issues in the Internet of Things for a smart city. Paper presented at: 2012 IEEE International Symposium on Communications and Information Technologies (ISCIT); 2012; Gold Coast, Australia.

42. Simmhan Y, Aman S, Kumhare A, et al. Cloud-based software platform for data-driven smart grid management. IEEE/AIP Comput Sci Eng. 2013;79.

43. Amin SM, Wollenberg BF. Toward a smart grid: power delivery for the 21st century. IEEE Power Energy Mag. 2005;3(5):34-41.

44. Aman S, Simmhan Y, Prasanna VK. Holistic measures for evaluating prediction models in smart grids. IEEE Trans Knowl Data Eng. 2015;27(2):475-488.

45. Aman S, Frincu M, Chelmis C, Noor M, Simmhan Y, Prasanna VK. Prediction models for dynamic demand response: requirements, challenges, and insights. Paper presented at: 2015 IEEE International Conference on Smart Grid Communications (SmartGridComm); 2015; Miami, FL. https://doi.org/10.1109/SmartGridComm.2015.7436323.

46. Mosleh K, Kumar R. A reliability perspective of the smart grid. IEEE Trans Smart Grid. 2010;1(1):57-64.

47. Dinh HT, Lee C, Niyato D, Wang P. A survey of mobile cloud computing: architecture, applications, and approaches. Wirel Commun Mob Comput. 2013;13(18):1587-1611.

48. Sanaei Z, Abolfazli S, Gani A, Buyya R. Heterogeneity in mobile cloud computing: taxonomy and open challenges. IEEE Commun Surv Tutor. 2014;16(1):369-392.

49. Fernando N, Loke SW, Rahayu W. Mobile cloud computing: a survey. Futur Gener Comput Syst. 2013;29(1):84-106.

50. Chun BG, Ihm S, Maniatis P, Naik M, Patti A. CloneCloud: elastic execution between mobile device and cloud. Paper presented at: 6th Conference on Computer Systems (EuroSys); 2011; Salzburg, Austria.

51. Barbera MV, Kosta S, Mei A, Stefa J. To offload or not to offload? The bandwidth and energy costs of mobile cloud computing. In: 2013 Proceedings IEEE INFOCOM; 2013; Turin, Italy.

52. Deng S, Huang L, Taheri J, Zomaya AY. Computation offloading for service workflow in mobile cloud computing. IEEE Trans Parallel Distributed Syst. 2015;26(12):3317-3329.

53. Xhafa F, Abraham A. Computational models and heuristic methods for grid scheduling problems. Futur Gener Comput Syst. 2010;26(4):608-621.

54. Dong F, Aki SG. Scheduling Algorithms for Grid Computing: State of the Art and Open Problems. Technical report. Kingston, Canada: Queen’s University; 2006.
55. Foster I, Zhao Y, Raicu I, Lu S. Cloud computing and grid computing 360-degree compared. Paper presented at: 2008 Grid Computing Environments Workshop; 2008; Austin, TX.

56. Abriashmi S, Naghibzadeh M, Epema DH. Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds. *Futur Gener Comput Syst.* 2013;29(1):158-169.

57. Imai S, Chestna T, Varela CA. Elastic scalable cloud computing using application-level migration. Paper presented at: 2012 IEEE 5th International Conference on Utility and Cloud Computing (UCC); 2012; Chicago, IL.

58. Raicu I, Foster IT, Zhao Y. Many-task computing for grids and supercomputers. Paper presented at: 2008 IEEE Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS); 2008; Austin, TX.

59. Topcuoglu H, Hariri S, Wu MY. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans Parallel Distributed Syst.* 2002;13(3):260-274.

60. Jackson KR, Muriki K, Ramakrishnan L, Runge KJ, Thomas RC. Performance and cost analysis of the supernova factory on the Amazon AWS cloud. *Sci Program Sci Driven Cloud Comput.* 2011;19(2-3):107-119.

61. Iosup A, Yigitbasi N, Epema D. On the performance variability of production cloud services. Paper presented at: 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing; 2011; Newport Beach, CA.

62. Mao M, Humphrey M. Auto-scaling to minimize cost and meet application deadlines in cloud workflows. Paper presented at: IEEE/ACM 11th International Conference on High Performance Computing, Networking, Storage and Analysis (SC); 2011; Seattle, WA.

63. Calheiros RN, Buyya R. Meeting deadlines of scientific workflows in public clouds with tasks replication. *IEEE Trans Parallel Distributed Syst.* 2014;25(7):1787-1796.

64. Jackson KR, Ramakrishnan L, Muriki K, et al. Performance analysis of high-performance computing applications on the Amazon web services cloud. Paper presented at: 2010 IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom); 2010; Indianapolis, IN.

65. Sandholm T, Gardfjäll P, Elmoth E, Johnsson L, Mulmo O. An OGSA-based accounting system for allocation enforcement across HPC centers. Paper presented at: ACM 2nd International Conference on Service Oriented Computing (ICSOC); 2004; New York, NY.

66. Brogi A, Forti S. QoS-aware deployment of IoT applications through the fog. *IEEE Internet Things J.* 2017;4(5):1185-1192.

67. Shi W, Cao J, Zhang Q, Li Y, Xu L. Edge computing: vision and challenges. *IEEE Internet Things J.* 2016;3(5):637-646.

68. Vaquero LM, Rodero-Merino L. Finding your way in the fog: towards a comprehensive definition of fog computing. *ACM SIGCOMM Comput Commun Rev.* 2014;44(5):27-32.

69. Mell P, Grance T. *The NIST Definition of Cloud Computing.* Gaithersburg, MD: National Institute of Standards and Technology; 2011.

70. Buyya R, Yeo CS, Venugopal S. Market-oriented cloud computing: vision, hype, and reality for delivering IT services as computing utilities. Paper presented at: 2008 10th IEEE International Conference on High Performance Computing and Communications; 2008; Dalian, China.

71. Luan TH, Gao L, Li Z, Xiang Y, Wei G, Sun L. Fog computing: focusing on mobile users at the edge. CoRR. 2015. https://arxiv.org/abs/1502.01815

72. Bittencourt LF, Lopes MM, Petri I, Rana OF. Towards virtual machine migration in fog computing. Paper presented at: 2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (PGCIC); 2015; Krakow, Poland.

73. Loke SW. The internet of flying-things: Opportunities and challenges with airborne fog computing and mobile cloud in the clouds. CoRR. 2015. https://arxiv.org/abs/1507.04492

74. Consortium OF. *OpenFog Reference Architecture.* Technical paper. 2018. https://www.openfogconsortium.org/ra/

75. Gedeon J, Meurisch C, Bhat D, Stein M, Wang L, Mühlhäuser M. Router-based brokering for surrogate discovery in edge computing. Paper presented at: 2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW); 2017; Atlanta, GA.

76. Saurez E, Hong K, Lilletun D, Ramachandran U, Ottenwärder B. Incremental deployment and migration of geo-distributed situation awareness applications in the fog. Paper presented at: ACM International Conference on Distributed and Event-Based Systems (DEBS); 2016; Irvine, CA.

77. Soo S, Chang C, Srirama SN. Proactive service discovery in fog computing using mobile ad hoc social network in proximity. Paper presented at: 2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData); 2016; Chengdu, China.

78. Rejiba Z, Masip-Bruin X, Jurnet A, Marin-Tordera E, Ren G-J. F2C-aware: enabling discovery in Wi-Fi-powered Fog-to-Cloud (F2C) systems. Paper presented at: 2018 6th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (Mobile-Cloud); 2018; Bamberg, Germany.

79. Truong NB, Lee GM, Ghamri-Doudane Y. Software defined networking-based vehicular adhoc network with fog computing. Paper presented at: 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM); 2015; Ottawa, Canada.

80. Aazam M, Huh EN. Fog computing and smart gateway based communication for cloud of things. Paper presented at: 2014 International Conference on Future Internet of Things and Cloud; 2014; Barcelona, Spain.

81. Yannuzzi M, Milito R, Serral-Gracià R, Montero D, Nemirovsky M. Key ingredients in an IoT recipe: fog computing, cloud computing, and more fog computing. Paper presented at: 2014 IEEE 19th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD); 2014; Athens, Greece.

82. Madsen H, Burschby B, Albeau G, Popentiu-Vlădicescu F. Reliability in the utility computing era: towards reliable fog computing. Paper presented at: 2013 20th IEEE International Conference on Systems, Signals and Image Processing (IWSSIP); 2013; Bucharest, Romania.

83. Yannuzzi M, van Lingen F, Jain A, et al. A new era for cities with fog computing. *IEEE Internet Comput.* 2017;21(2):54-67.
84. Dastjerdi AV, Buyya R. Fog computing: helping the Internet of Things realize its potential. *Computer*. 2016;49(8):112-116. https://doi.org/10.1109/MC.2016.245

85. García Lopez P, Montresor A, Epema D, et al. Edge-centric computing: vision and challenges. *ACM SIGCOMM Comput Commun Rev*. 2015;45(5):37-42.

86. Satyanarayanan M, Simoes P, Xiao Y, et al. Edge analytics in the Internet of Things. *IEEE Pervasive Comput*. 2015;14(2):24-31.

87. Malawski M, Juve G, Deelman E, Nabrzyski J. Cost- and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds. Paper presented at: IEEE/ACM International Conference on High Performance Computing, Networking, Storage and Analysis (SC); 2012; Salt Lake City, UT.

88. Chu HY, Simmhan Y. Cost-efficient and resilient job life-cycle management on hybrid clouds. Paper presented at: 2014 IEEE 28th International Parallel and Distributed Processing Symposium (IPDPS); 2014; Phoenix, AZ.

89. Poola D, Ramamohanarao K, Buyya R. Fault-tolerant workflow scheduling using spot instances on clouds. *Procedia Comput Sci*. 2014;29:523-533.

90. OpenStack. 2018. https://www.openstack.org/

91. Burns B, Grant B, Oppenheimer D, Brewer E, Wilkes J. Borg, omega, and Kubernetes. *Queue*. 2016;14(1):10.

92. Yi S, Hao Z, Qin Z, Li Q. Fog computing: platform and applications. Paper presented at: 2015 3rd IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb); 2015; Washington, DC.

93. Lebre A, Pastor J, Simonet A, Desreuz F. Revising OpenStack to operate fog/edge computing infrastructures. Paper presented at: 2017 IEEE International Conference on Cloud Engineering (IC2E); 2017; Vancouver, Canada.

94. Chang H, Hari A, Mukherjee S, Lakshman T. Bringing the cloud to the edge. Paper presented at: 2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS); 2014; Toronto, Canada.

95. OpenStack. Fog edge massively distributed clouds. 2019. https://wiki.openstack.org/wiki/Fog_Edge_Massively_Distributed_Clouds

96. Tsai PH, Hong HJ, Cheng AC, Hsu CH. Distributed analytics in fog computing platforms using tensorflow and Kubernetes. Paper presented at: 2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS); 2017; Seoul, South Korea.

97. Hong HJ, Tsai PH, Hsu CH. Dynamic module deployment in a fog computing platform. Paper presented at: 2016 18th Asia-Pacific Network Operations and Management Symposium (APNOMS); 2016; Kanazawa, Japan.

98. Eclipse. 2019. https://www.eclipse.org/kura/

99. Bellavista P, Zanni A. Feasibility of fog computing deployment based on docker containerization over raspberrypi. In: Proceedings of the 18th International Conference on Distributed Computing and Networking (ICDCN); 2017; Hyderabad, India.

100. The Linux Foundation Projects. EdgeXFoundry: the open platform for the IoT edge. 2019. https://www.edgexfoundry.org/

101. Zhou AC, He B. Transformation-based monetary cost optimizations for workflows in the cloud. *IEEE Trans Cloud Comput*. 2014;2(1):85-98.

102. Agmon Ben-Yehuda O, Ben-Yehuda M, Schuster M, Tsafir D. Deconstructing Amazon EC2 spot instance pricing. *ACM Trans Econ Comput*. 2013;1(3):16:1-16:20.

103. Chohan N, Castillo C, Spreitzer M, Steinbauer M, Tantawi A, Krintz C. See spot run: using spot instances for MapReduce workflows. Paper presented at: USENIX Conference on Hot Topics in Cloud Computing; 2010; Boston, MA.

104. Kushwaha V, Simmhan Y. Cloudy with a spot of opportunity: analysis of spot-priced VMs for practical job scheduling. Paper presented at: 2014 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM); 2014; Bangalore, India.

105. Voorsluys W, Buyya R. Reliable provisioning of spot instances for compute-intensive applications. Paper presented at: 2012 IEEE 26th International Conference on Advanced Information Networking and Applications (AINA); 2012; Fukuoka, Japan.

106. Subramanya S, Guo T, Sharma P, Irwin D, Shenoy P. SpotOn: a batch computing service for the spot market. Paper presented at: 6th ACM Symposium on Cloud Computing (SoCC); 2015; Kohala Coast, HI.

107. Bittencourt LF, Madeira ERM. HCOC: a cost optimization algorithm for workflow scheduling in hybrid clouds. *J Internet Serv Appl*. 2011;2(3):207-227.

108. Guo T, Sharma U, Shenoy P, Wood T, Sahu S. Cost-aware cloud bursting for enterprise applications. *ACM Trans Internet Technol*. 2014;13(3):10:1-10:24.

109. Dörnemann T, Juhnke E, Freisleben B. On-demand resource provisioning for BPEL workflows using Amazon’s elastic compute cloud. Paper presented at: 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID); 2009; Shanghai, China.

110. Poola D, Ramamohanarao K, Buyya R. Enhancing reliability of workflow execution using task replication and spot instances. *ACM Trans Auton Adapt Syst*. 2016;10(4):30:1-30:21.

111. Miettinen AP, Nurminen JK. Energy efficiency of mobile clients in cloud computing. In: Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing (HotCloud); 2010; Boston, MA.

112. Misra P, Simmhan Y, Warrior J. Towards a practical architecture for india centric Internet of Things. *IEEE IoT Newsletter*. 2015.

113. Thai L, Varghese B, Barker A. Task scheduling on the cloud with hard constraints. Paper presented at: 2015 IEEE World Congress on Services; 2015; New York, NY.

114. Sakellariou R, Zhao H, Tsiakkouri E, Dikaiakos MD. Scheduling workflows with budget constraints. In: *Integrated Research in GRID Computing: CoreGRID Integration Workshop 2005 (Selected Papers) November 28-30, Pisa, Italy*. Boston, MA: Springer Science+Business Media; 2007:189-202.

115. Badiger S, Baheti S, Simmhan Y. VioLET: a large-scale virtual environment for Internet of Things. Paper presented at: International European Conference on Parallel Processing (EuroPar); 2018; Turin, Italy. https://github.com/dream-lab/VioLET
116. Rodriguez MA, Buyya R. Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds. *IEEE Trans Cloud Comput*. 2014;2(2):222-235.

117. Chen W, Lee YC, Fekete A, Zomaya AY. Adaptive multiple-workflow scheduling with task rearrangement. *J Supercomput*. 2015;71(4):1297-1317.

118. Kumbhare AG, Simmhan Y, Prasanna VK. PLASiCC: predictive look-ahead scheduling for continuous dataflows on clouds. Paper presented at: 2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid); 2014; Chicago, IL.

119. Vishwanath KV, Nagappan N. Characterizing cloud computing hardware reliability. Paper presented at: 1st ACM Symposium on Cloud Computing (SoCC); 2010; Indianapolis, IN.

120. Simmhan Y, Barga R, van Ingen C, Lazowska E, Szalay A. On building scientific workflow systems for data management in the cloud. Paper presented at: 2008 IEEE 4th International Conference on eScience (eScience); 2008; Indianapolis, IN.

121. Varshney P, Simmhan Y. AutoBot: resilient and cost-effective scheduling of a bag of tasks on spot VMs. *IEEE Trans Parallel Distributed Syst*. 2018. https://doi.org/10.1109/TPDS.2018.2899851

122. Wu Z, Liu X, Ni Z, Yuan D, Yang Y. A market-oriented hierarchical scheduling strategy in cloud workflow systems. *J Supercomput*. 2013;63(1):256-293.

123. Callaghan S, Maechling P, Small P, et al. Metrics for heterogeneous scientific workflows: a case study of an earthquake science application. *Int J High Perform Comput Appl*. 2011;25(3):274-285.

124. Deelmaen E, Singh G, Livny M, Berriman B, Good J. The cost of doing science on the cloud: the montage example. In: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing (SC); 2008; Austin, TX.

125. Abramson D, Giddy J, Kotler L. High performance parametric modeling with Nimrod/G: killer application for the global grid? Paper presented at: 14th IEEE International Parallel and Distributed Processing Symposium (IPDPS); 2000; Cancun, Mexico.

126. Maheswaran M, Ali S, Siegel HJ, Hensgen D, Freund RF. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. Paper presented at: 8th IEEE Heterogeneous Computing Workshop (HCW); 1999; San Juan, Puerto Rico.

127. Oprescu A-M, Kielmann T. Bag-of-tasks scheduling under budget constraints. Paper presented at: 2010 IEEE 2nd International Conference on Cloud Computing Technology and Science (CloudCom); 2010; Indianapolis, IN.

128. Xu M, Cui L, Wang H, Bi Y. A multiple QoS constrained scheduling strategy of multiple workflows for cloud computing. Paper presented at: 2009 IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA); 2009; Chengdu, China.

129. Simmhan Y, van Ingen C, Szalay A, Barga R, Heasley J. Building reliable data pipelines for managing community data using scientific workflows. Paper presented at: 2009 5th IEEE International Conference on e-Science (e-Science); 2009; Oxford, UK.

130. Yi S, Kondo D, Andrezejak A. Reducing costs of spot instances via checkpointing in the Amazon elastic compute cloud. Paper presented at: 2010 IEEE International Conference on Cloud Computing; 2010; Miami, FL.

131. Jung D, Lim J, Yu H, Suh T. Estimated interval-based checkpointing (EIC) on spot instances in cloud computing. *J Appl Math*. 2014;2014:217547:1-217547:12.

132. Kim H, el Khamra Y, Roderio I, Jha S, Parashar M. Autonomic management of application workflows on hybrid computing infrastructure. *Scientific Programming*. 2011;19(2-3):75-89.

133. Aazam M, St-Hilaire M, Lung C-H, Lambadaris I. MeFoRE: QoE based resource estimation at Fog to enhance QoS in IoT. Paper presented at: 2016 23rd International Conference on Telecommunications (ICT); 2016; Thessaloniki, Greece.

134. Garg SK, Buyya R, Siegel HJ. Scheduling parallel applications on utility grids: time and cost trade-off management. Paper presented at: 32nd Australasian Conference on Computer Science (ACSC); 2009; Wellington, New Zealand.

135. Feng H, Song G, Zheng Y, Xia J. A deadline and budget constrained cost-time optimization algorithm for scheduling dependent tasks in grid computing. Paper presented at: International Conference on Grid and Cooperative Computing (GCC); 2003; Shanghai, China.

136. Fard HM, Frodan R, Fähringer T. A truthful dynamic workflow scheduling mechanism for commercial multicloud environments. *IEEE Trans Parallel Distributed Syst*. 2013;24(6):1203-1212.

137. Wang S, Yu X, Yu X. Survivability-based scheduling algorithm for bag-of-tasks applications with deadline constraints on grids. *Int J Comput Sci Netw Secur*. 2006;6(4):13-18.

138. Liu K, Jin H, Chen J, Liu X, Yuan D, Yang Y. A compromised-time-cost scheduling algorithm on a cloud computing platform. *Int J High Perform Comput Appl*. 2010;24(4):445-456.

139. Diaz-Montes J, Diaz-Granados M, Zou M, Tao S, Parashar M. Supporting data-intensive workflows in software-defined federated multi-clouds. *IEEE Trans Cloud Comput*. 2015;6(1):250-263.

140. Yu J, Buyya R. A taxonomy of scientific workflow systems for grid computing. *ACM SIGMOD Rec*. 2005;34(3):44-49.

141. Liu L, Zhang M, Lin Y, Qin L. A survey on workflow management and scheduling in cloud computing. Paper presented at: 2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing; 2014; Chicago, IL.

142. Yamada T, Nakano R. Job shop scheduling. In: *Genetic Algorithms in Engineering Systems*. Chapter 7. Stevenage, UK: The Institution of Electrical Engineers; 1997:134-160.

143. Convolbo MW, Chou J. Cost-aware DAG scheduling algorithms for minimizing execution cost on clouds. *J Supercomput*. 2016;72(3):985-1012.

144. Ghosh R, Simmhan Y. Distributed scheduling of event analytics across edge and cloud. *ACM Trans Cyber Phys Syst*. 2018;2(4):24:1-24:28.

145. Zheng W, Sakellariou R. Budget-deadline constrained workflow planning for admission control. *J Grid Comput*. 2013;11(4):633-651.

146. Lin C, Lu S. Scheduling scientific workflows elastically for cloud computing. Paper presented at: 2011 IEEE 4th International Conference on Cloud Computing (CLOUD); 2011; Washington, DC.
176. Ananthanarayanan G, Bahl P, Bodik P, et al. Real-time video analytics: the killer App for edge computing. *Computer*. 2017;50(10):58-67.

177. Wang J, Amos B, Das A, Pillai P, Sadeh N, Satyanarayanan M. A scalable and privacy-aware IoT service for live video analytics. In: Proceedings of the 8th ACM on Multimedia Systems Conference (MMSys); 2017; Taipei, Taiwan. http://doi.acm.org/10.1145/3083187.3083192

178. Lo V, Zappala D, Zhou D, Liu Y, Zhao S. Cluster computing on the fly: P2P scheduling of idle cycles in the Internet. In: *Peer-to-Peer Systems III: Third International Workshop, IPTPS 2004, La Jolla, CA, USA, February 26-27, 2004, Revised Selected Papers*. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 2005:227-236.

**How to cite this article:** Varshney P, Simmhan Y. Characterizing application scheduling on edge, fog, and cloud computing resources. *Softw: Pract Exper*. 2020;50:558–595. https://doi.org/10.1002/spe.2699
FIGURE A1  Complete taxonomy as a single tree [Colour figure can be viewed at wileyonlinelibrary.com]