EDITOR: Schahram Dustdar, dustdar@infosys.tuwien.ac.at

Serverless Edge Computing—Where We Are and What Lies Ahead

Philipp Raith, Stefan Nastic, and Schahram Dustdar, *TU Wien, Wien, 1040, Austria*

The edge–cloud continuum combines heterogeneous resources, which are complex to manage. Serverless edge computing is a suitable candidate to manage the continuum by abstracting away the underlying infrastructure, improving developers' experiences, and optimizing overall resource utilization. However, understanding and overcoming programming support, reliability, and performance engineering challenges are essential for the success of serverless edge computing. In this article, we review and evaluate the maturity of serverless approaches for the edge–cloud continuum. Our review includes commercial, community-driven offerings and approaches from academia. We identify several maturity levels of serverless edge computing and use them as criteria to evaluate the maturity of current state-of-the-art serverless approaches with a special focus on the programming, reliability, and performance challenges. Finally, we lay a road map toward the next generation of serverless edge computing systems.

The edge–cloud continuum is a heterogeneous infrastructure of computing resources ranging from handheld devices to server-grade hardware. The combination of emerging application paradigms, such as edge intelligence (EI),[1] with increasingly complex requirements and the heterogeneous infrastructure poses new challenges for managing, scaling, and deploying applications. Current cloud-centric platforms are not specifically designed for these heterogeneous environments. Serverless edge computing promises to efficiently use resources and reduce costs by abstracting the infrastructure and application management (e.g., scaling) away from users.[2,3] Unfortunately, current serverless edge computing approaches still face numerous challenges to unlock the full potential of the edge–cloud continuum.[2]

To gain a deeper understanding of serverless platforms, we review the current state of serverless edge computing. This review is important to see where the serverless community, including commercial providers, open source projects, and academia, currently is and which challenges remain. We present a set of criteria that define maturity levels to systematically compare and evaluate the maturity of current serverless approaches to achieve the vision of a unified serverless computing fabric (SCF) for the edge–cloud continuum.[4] The SCF unites storage and compute resources in the edge–cloud continuum and enables new application paradigms.[4]

One of those is EI, which promises to enable new use cases (e.g., mobile augmented reality and autonomous vehicles) by distributing artificial intelligence (AI) inference and training across the continuum.[1] We identify three main challenges that the SCF must address:

- EI applications are complex and span across the continuum. They range from short-running inference to long-running and data-intensive training tasks. Developing these applications is challenging, and the SCF must provide proper programming support to simplify the development process.
- Mission-critical EI applications (e.g., emergency rescue) and long-running and data-intensive tasks (e.g., training) require reliable execution to guarantee service and save costs. It is challenging to implement and enable these applications in the dynamic and heterogeneous environment of the edge–cloud continuum.

1089-7801 © 2023 IEEE
Digital Object Identifier 10.1109/MIC.2023.3260939
Date of current version 9 May 2023.

› EI applications run on highly heterogeneous hardware, and workloads can highly fluctuate (e.g., a public event that draws a large crowd). The SCF must address the challenge of providing optimal performance under these conditions.

Based on these challenges, our evaluation focuses on evaluating the maturity of 1) programming support, 2) reliability engineering, and 3) performance engineering the serverless edge computing.

MATURITY LEVELS OF SERVERLESS EDGE COMPUTING

We define maturity levels to evaluate serverless platforms and highlight key aspects for serverless edge computing platforms. Our criteria are split into *design time* and *runtime* (see Figure 1). The *design time* deals with tasks that happen before deploying the function. The *runtime* focuses on challenges that happen after the deployment.

Design-Time Criteria

Application State and Data Management

The application state and data management considers design-time mechanisms to support developers in writing stateful functions or accessing external storage (e.g., object storage). We introduce a three-level maturity rating system that evaluates if the programming model or platform [maturity level (M1)] offers *any functionality*, such as checkpointing, to solve data-related issues (e.g., statefulness); (M2) incorporates data management as a *first-class citizen* [e.g., by introducing suitable application programming interface (API) abstractions to facilitate application state and data management], and (M3) offers capabilities to handle data management transparently to developers (e.g., *failure handling*).

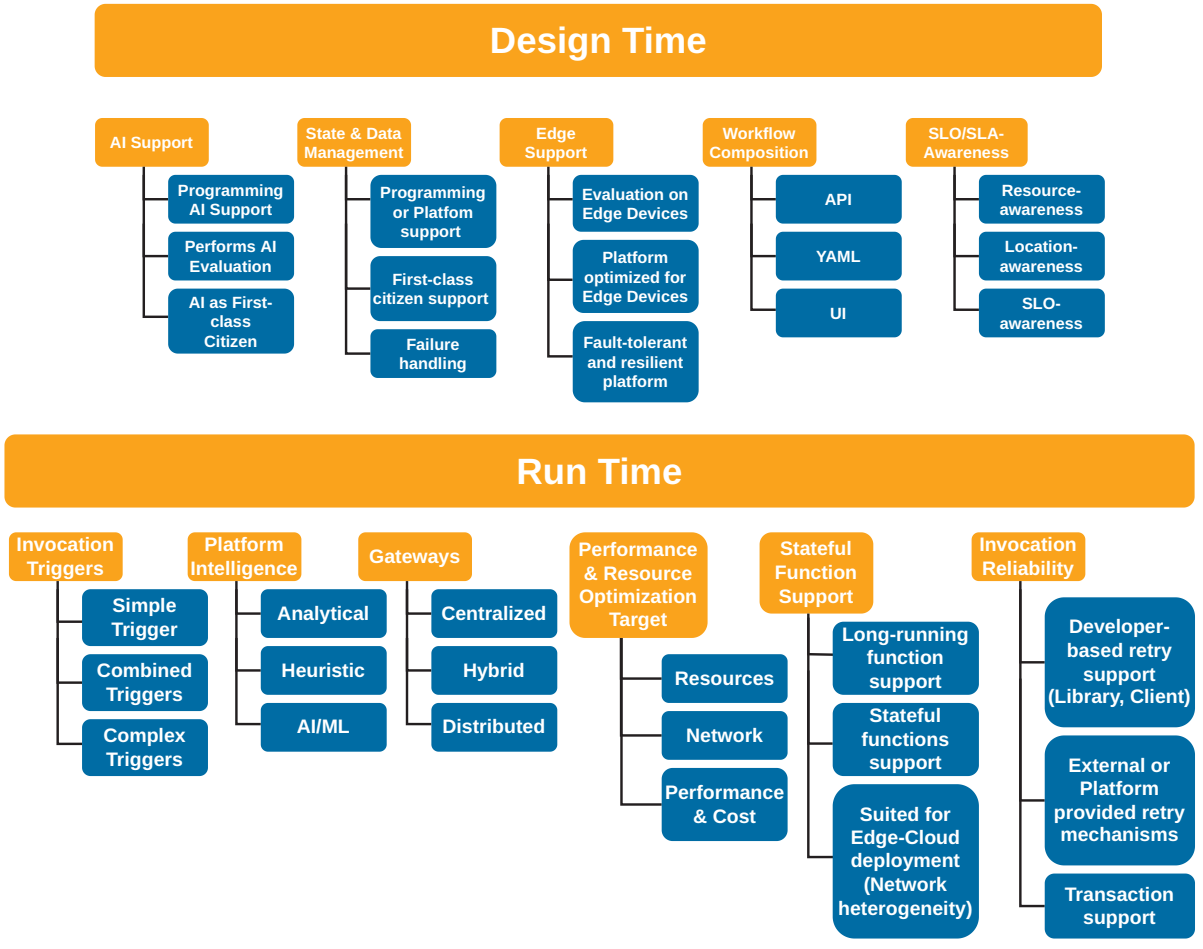

FIGURE 1. Design-time and runtime maturity levels. AI: artificial intelligence; API: application programming interface; ML: machine learning; SLO: service level objective; UI: user interface.

SLO/SLA Awareness

Service level objective (SLO)/service level agreement (SLA) awareness allows developers to specify requirements and constraints during design time. The heterogeneous infrastructure and stringent application requirements make it necessary that developers are able to express requirements.[5,6] We define a three-level maturity rating that ranks different types of SLOs and SLAs based on the complexity of the requirement. M1 specifies the *resource awareness*: whether resource consumption is considered and optimized. Approaches that reduce resource consumption (i.e., a lightweight runtime, reduced network traffic, etc.) or offer resource-oriented SLOs fulfill this level. M2 specifies the *location awareness*: whether developers can express location requirements. M3 specifies the *SLO awareness*: whether developers can influence the execution of functions. Serverless platforms often allow developers to specify the number of available cores or memory, but we think the SCF has to offer, in addition, more sophisticated features (e.g., latency requirements[7]).

AI Support

We define three levels to evaluate the maturity of AI support. M1 specifies the *support of programming languages* that at least offer AI frameworks (e.g., Python). M2 requires that at least *AI workloads have been considered during the evaluation*. M3 is treating AI applications as *first-class citizens*. For example, the programming model defines specific abstractions tailored to AI applications.

Edge Support

Edge support shows the maturity of functions to run across the continuum. M1 shows that real-world edge devices (e.g., a Raspberry Pi, embedded AI devices, or small-form computers) have been *used during evaluation or development*. M2 specifies if the platform is *optimized for resource-constrained edge devices*. These optimizations can include a low resource footprint, reduced network traffic, energy awareness, or optimized performance on resource-constrained devices. M3 specifies that the support for *fault tolerance and reliability in edge–cloud systems* has been explicitly considered.[8]

Workflow Composition

Workflow composition enables developers to build complex applications consisting of multiple functions. Workflow composition can be done via a user interface (UI) (i.e., AWS Step Functions[a])—developers can drag and drop functions and connect inputs and outputs. A UI can reduce the entry barrier by offering a convenient solution. Developers can also compose functions in code (i.e., the API). Auxiliary deployment files (i.e., YAML) or domain-specific languages (DSLs) can also compose functions and decouple the definition from the business logic. We introduce the three-level maturity system and rank the approaches by the ease of use: (M1) *API* (in code), (M2) *auxiliary metadata files*, and (M3) *UI*.

Runtime Criteria

Invocation Reliability

Invocation reliability specifies the fault tolerance and reliability of function invocations. We introduce a three-level maturity system. M1 specifies that *developers must take care* of failed invocations (e.g., by manual retries). M2 specifies the support of *retry-based invocation mechanisms*. Platform-based retry mechanisms help mitigate faults during runtime and are fundamental to at-least-once semantics.[9] M3 specifies *transaction support*, which guarantees graceful handling of errors in workflows.[10] For example, platforms may offer support to compose workflows that are able to react to failures and execute recovery actions.

Stateful Functions

Stateful and long-running functions are becoming increasingly relevant.[11] Serverless platforms were initially designed for short-running tasks (under 15 min), and many current serverless offerings still have strict timeouts. Therefore, we introduce a three-level maturity evaluation to determine the level of support for executing stateful functions. M1 specifies the possibility of performing *long-running functions*. M2 specifies the explicit *support for stateful functions* and checkpointing. M3 specifies if the platform has been *tested or implemented with the edge–cloud* in mind—specifically, if the platform addresses heterogeneous network conditions or node capacities.

Invocation Triggers

Invocation triggers are invoked by clients to execute functions. Serverless platforms usually support multiple types: HTTP, queue, and other types of events (e.g., file changes).[12] Therefore, we introduce a three-level system to evaluate the maturity of triggers. M1 means that *only one trigger type* is available (i.e., HTTP or queue), M2 means that *HTTP and queue triggers* are considered, and M3 specifies that the platform allows *complex triggers* composed of multiple event sources (e.g., Amazon EventBridge[b]).

[a]https://aws.amazon.com/step-functions/

[b]https://aws.amazon.com/eventbridge/

Edge–Cloud Gateways

Edge–cloud gateways serve as entry points for clients to invoke and route requests to function instances. Current cloud-centric platforms deploy *centralized* gateways, which incur high network latency and practically diminish the value of deployed hardware at the edge. We introduce three levels to evaluate the maturity of gateways. M1 specifies that only a *centralized* gateway is available (i.e., cloud-centric). M2 is a *hybrid* approach in which the platform deploys multiple gateways across the infrastructure. The gateways can be dynamically placed and scaled but are commonly deployed at the network's edge (e.g., 5G base stations). M3 is the *distributed* approach, in which each node acts as a gateway and can route requests.

Platform Intelligence

Platform intelligence describes how smart or intelligent the typical runtime mechanisms (e.g., scaling, placement, and routing) are: (M1) *analytical*, (M2) *heuristic-based*, and (M3) *AI/machine learning (ML)-driven* approaches. The platform must adapt to the changing infrastructure and know the current state (e.g., capacities and workload) in the edge–cloud continuum. Analytical solutions (e.g., the *target value/current value*) are not resource intensive and do not capture intricate details to optimize functions. *Heuristic-based platform intelligence* relies on optimization processes and is able to capture important details and optimize toward multiple goals[13]). *AI/ML-driven platform intelligence* offers methods, such as reinforcement learning, to continuously learn and optimize the function deployments.

Performance and Resource Consumption Optimization Targets

Different serverless platforms are designed to be optimized for a specific target of function execution. Such targets can range from low latency to cost efficiency or resource usage. We introduce a three-level maturity system that ranks the *optimization target* by platform customer relevancy: M1 means that the *optimization target* targets *resources* (e.g., the number of function instances). Customers usually pay for the execution time and do not care about the used *resources*. M2 means that the optimization targets the *network*. For example, function instances placed far apart can lead to expensive cross-region network traffic and incur high latency. M3 means the platform can optimize for *performance or cost* when executing serverless functions in the edge–cloud continuum.

WHERE ARE WE NOW?

Our review includes more than 45 commercial, open source platforms and academic works. We include only platforms that are actively maintained, meaning they are not explicitly archived or abandoned.[c] Further, we include only works that at least have evaluated a prototype and, thus, ignore others that propose concepts or simulation-only evaluations. To structure the review, we group the approaches into four categories:

- *Core serverless platforms* include bare-bones platforms to autonomously manage functions. This category includes platforms like AWS Lambda,[15] OpenFaaS,[16] and from prototypes from academia.[17,18]
- *Serverless platform extensions* include research works that improve existing platforms by introducing novel scaling, placement, or routing techniques.[19,20]
- *Serverless programming extensions* include open source projects, such as Zappa[21] or Chalice,[22] and research works that introduce new approaches of programming serverless functions.[23,24]
- *Serverless function runtimes* focus on the execution aspect of functions and mostly consist of works that introduce new virtualization or isolation techniques.[25,26]

Tables 1 and 2 show the results of our literature review for the design time and runtime, respectively. In the following, we structure our key insights based on the challenges we previously identified for the SCF.[4] Specifically, we discuss the following topics:

- *Programming support* is specifically evaluated based on the available support for *AI support* and *state and data management* at design time.
- *Reliability engineering* is evaluated based on *SLO support* and *reliable invocations*.
- *Performance and infrastructure engineering* is evaluated based on *optimization* and *edge support*.

Programming Support

AI Support

Our review shows that only two open source platforms focus explicitly on AI and that two commercial platforms provide AI-focused services: KServe,[27] which extends Knative,[28] and Nuclio.[29] KServe focuses on AI inference and supports developers in deploying

[c]For example, we omit Kubeless, a popular open source platform, because it is currently archived on GitHub and not maintained.[14]

inference functions by supporting production-ready inference serving engines (e.g., TFServing[d]).

Nuclio[29] focuses on data pipelines, including training. Nuclio offers an optimized function runtime that enables it to offer high parallelism and a minimized footprint.[30] *Serverless platform extensions* treat AI as first-class citizen[31] by letting developers specify required AI models that the placement component considers to reduce network traffic. Only one reviewed work[32] does not support languages with AI frameworks (i.e., they only support Lua). Others have evaluated AI workloads, but only a few treat them as first-class citizens, as can be seen in Table 1. For example, Wang et al.[33] deploy AI functions on edge hardware and perform workload experiments to evaluate their scaling and placement components. Lordan et al.[34] perform similar experiments to evaluate their platform and resource-constrained devices. Specifically, Google Cloud Functions and AWS Greengrass provide services to ease the development and deployment of AI workloads.

Table 2 shows that most commercial platforms currently do not offer any hardware accelerator support, and open source ones support them only to a certain degree. The only exception of commercial platforms is AWS Greengrass, which lets developers mount and use hardware accelerators in functions. Further, we consider platforms that run on top of Kubernetes to support hardware accelerators because Kubernetes allows the mounting of GPUs and other hardware accelerators into functions. Interestingly, the work by Hu et al.[35] investigates hardware accelerators to improve input–output operations instead of compute operations. Werner and Schirmer[18] introduce a novel level of abstraction in which the platform autonomously chooses the appropriate accelerator. AI serverless platforms (i.e., KServe and Nuclio) offer hardware accelerator support, and two serverless extensions[20,36] explore options to enable remote GPU sharing and GPUs in edge–cloud scenarios, respectively. We can see, in Table 2, that none of the *serverless function runtimes* consider hardware accelerators and that two *serverless platform extensions* and only eight *core serverless platforms* support it.

Application State and Data Management

Application state and data management is prevalent across application paradigms, and, especially, AI applications tend to be data intensive[31] and stateful.[37] Table 1 shows that *serverless programming extensions* can help increase the maturity of *state and data management* (see Figure 1), but only few platforms focus on it. For example, Karhula et al.[38] focus on enabling seamless checkpointing for function migrations. Kappa[24] introduces new Python programming primitives to take checkpoints and manage concurrency. This framework can be used to recover from failures during long-running training tasks. Rausch et al.[31] introduce annotations that help the placement component to reduce the function execution and lower network transfer by shifting the computation close to the data.

Others aim to abstract the storage layer by implementing high-level APIs for developers[10,35] or extend platforms by introducing actors for stateful computation[23,39] or by implementing an extra layer (e.g., shim) to handle storage operations.[9]

Table 2 shows that *stateful function support* is immature across all categories. Most of the commercial platforms do not offer long-running functions, while open source platforms generally can be configured with long timeouts. AWS Greengrass[40] allows long-running functions that can write and read to a device's storage and reuse variables. Cloudflare[e] offers *durable objects*, which give workers, located in CDNs, a unique ID and enables stateful computation. Shillaker and Pietzuch[26] implement a WebAssembly-based function runtime that implements an efficient local state access approach and enables stateful functions.

Reliability Engineering

Dynamicity and heterogeneity are key characteristics of the edge–cloud continuum. Serverless platforms have to address these challenges by providing resilient and fault-tolerant execution. Specifically, we investigate the maturity of platforms in terms of their support for specifying SLOs and how reliable function invocations are.

SLO/SLA

The *SLO awareness* (see Figure 1) maturity level is fulfilled by only a few approaches, and no commercial or open source platforms allow users to configure complex SLOs (e.g., latency requirements). However, most of those platforms support the configuration in terms of number of available CPU cores and memory. Klingler et al.[41] present a set of code annotations that allow developers to fine-tune the function execution. Specifically, these annotations can find the optimal function resource configuration (e.g., available memory) or avoid cold starts. Commercial and open source platforms partially support *location awareness*. Amazon and Microsoft offer edge-only environments to execute functions

[d]https://www.tensorflow.org/tfx/guide/serving

[e]https://developers.cloudflare.com/workers/learning/using-durable-objects/

and include custom edge devices that can interact with the cloud platforms [AWS Greengrass and Microsoft's Internet of Things (IoT) Edge]. Others let developers programmatically specify the location of function execution.[31,35,42] Hu et al.[35] implement a DSL that enables developers to specify the place of execution (i.e., Edge or Cloud). Smith et al.[42] allow clients to specify the location of function execution in the invocation request, while Rausch et al.[31] implement a location- and data-aware placement approach by enabling developers to specify the data (i.e., buckets of object storage) needed for function execution. The *resource awareness* maturity is fulfilled by reducing the network traffic[31,42] or resource footprint,[16,17,43–45] improving data exchange between function instances,[35,46] reducing the overhead of function runtimes,[27,29,40] or explicitly considering resource usage while scaling and placing serverless functions.[33,47]

Reliable Function Invocations

Reliable function invocations are essential due to dynamic device nature and heterogeneous networks in edge–cloud systems. Therefore, we evaluate the maturity of *invocation triggers*, *invocation reliability*, and *edge–cloud gateways* (see Figure 1). First, most of the *core serverless platforms* support HTTP, and all open source and commercial offerings provide additional queuing triggers. HTTP is the most stable and fastest method in public cloud offerings, enabling high-throughput EI applications (e.g., inference).[12] Queuing triggers can guarantee at-least-once execution semantics. Note that open source platforms typically rely on external queuing systems to enable retry mechanisms (e.g., NATS in OpenFaaS[16] and Kafka in Knative[28]). Using external queue mechanisms enhances the interoperability between platforms. In contrast, commercial platforms offer managed queuing solutions, contributing to the vendor lock-in effect.

Second, approaches offering queuing mechanisms fulfill retry-based *invocation reliability*. Only commercial platforms support mature *invocation reliability*[15,48,49] by using managed queueing systems to guarantee retries and allowing workflow transactions that can react to failures. One research paper[45] is also mature by using a distributed fault-tolerant, append-only storage. As shown in Table 1, several *serverless programming extensions* focus on fault tolerance.[9,10,24] These approaches add fault tolerance by 1) adding a coordinator to orchestrate the transactions,[24] 2) having a platform-independent runtime and a library that enables transactions on stateful workflows that are either all committed or reverted,[10] or 3) by introducing an extra layer between the platform and the storage to buffer updates and flush them upon successful workflow completion.[9]

Third, hybrid edge–cloud gateways are implemented by many approaches from the research community but lack adoption from open source projects, such as those by Baresi and Quattrocchi,[13] Hetzel et al.,[32] Lordan et al.,[34] and Baresi et al.[36] For example, some approaches implement decentralized edge clusters that act as the gateway and router.[13,34,36] Bermbach et al.[44] implement a *distributed edge–cloud gateway* by letting each node decide whether to execute the request locally or offload it to the next one until it reaches the cloud. Hu et al.[35] also implement the *distributed* approach by compiling a user-supplied workflow and specifying for each function in the workflow where the request should be executed (i.e., locally or cloud). Amazon's AWS Greengrass and Microsoft Azure IoT allow customers to set up edge clusters that can act as *hybrid edge–cloud gateways*. For example, Das et al.[50] dynamically decide whether to process an invocation request on the local AWS Greengrass deployment or offload the task to AWS Lambda.

IBM Cloud Functions have an edge extension that allows developers to pre- and postprocess requests at the Content Delivery Network (CDN) level, which might be considered a hybrid approach. Table 1 shows that only few support the *workflow composition* of functions via a UI (most notably commercial offerings) and based on metadata files (i.e., YAML). Most approaches allow the composition via the code API. Hu et al.[35] show the DSL with which developers can build application graphs and specify which functions can be run in parallel.

Performance and Infrastructure Engineering

Optimization of function execution is an important aspect in serverless edge computing, and platforms offer *performance and resource optimization targets* based on *platform intelligence* (see Figure 1). Table 2 shows that commercial approaches do not expose any *optimization targets* to users and few details on scaling and placement. Open source platforms use analytical approaches and, most often, reuse scaling methods from the container orchestration service (i.e., the default autoscaler from Kubernetes) or black-box performance metrics (i.e., requests per second) to determine the number of required function instances analytically. Further, all open source platforms solely offer *optimization targets* for scaling functions and do not offer any smart placement or routing mechanisms, except for round-robin or capacity-based ones.[28] However, works from academia offer different *optimization targets* and can offer mature *platform intelligence* solutions. *Core serverless platforms* and *serverless platform extensions* allow the specification of latency or deadline requirements using *optimization-based platform intelligence*,[33,36] reduce cost

TABLE 1. Design time.*

Reference	S & D	Workflow	Edge Support	AI Support	OSS	SLO/SLA Awareness	Deployment
Core serverless platforms							
AWS Lambda[15,†]	★☆☆	UI and API	☆☆☆	★☆☆	✗	Resource awareness	Managed
AWS Greengrass[40,†]	★☆☆	UI	★★★	★★★	✓	Resource and location awareness	Managed
Google Cloud Functions[48,†]	☆☆☆	API	☆☆☆	★★★	✗	Resource awareness	Managed
Microsoft Azure Functions[49,†]	★★★	UI and API	★★☆	★☆☆	✗	Resource and location awareness	Managed
IBM Cloud Functions[53,†]	☆☆☆	API	★☆☆	★☆☆	✗	Resource awareness	Managed
Apache OpenWhisk[43,‡]	☆☆☆	API	★★☆	★☆☆	✓	Resource and SLO awareness	Kubernetes
OpenFaaS[16,‡]	☆☆☆	API	★☆☆	★☆☆	✓	Resource and SLO awareness	Kubernetes
Knative[28,]	☆☆☆	API	★☆☆	★☆☆	✓	SLO awareness	Kubernetes
KServe[27,‡]	★★☆	YAML	★☆☆	★★★	✓	Resource and SLO awareness	Knative
Nuclio[29,‡]	★☆☆	API	★☆☆	★★★	✓	Resource awareness	Kubernetes/managed
Fission[54,‡]	★☆☆	API	☆☆☆	★☆☆	✓	Resource and SLO awareness	Kubernetes
Lordan et al.[34]	☆☆☆	API	★☆☆	★★☆	✗	Resource, location, and SLO awareness	Custom
Wolski et al.[45]	★★★	N/A	★★★	★☆☆	✓	Resource awareness	Custom
Smith et al.[42]	★★★	N/A	★★★	★☆☆	✓	Resource and location awareness	Multi-FaaS
Pfandzelter and Bermbach[17]	☆☆☆	N/A	★★☆	★☆☆	✓	Resource awareness	Custom
Akkus et al.[52]	★★★	API	★★★	★☆☆	✗	Resource awareness	Custom
Hu et al.[35]	★★★	API	★★★	★★☆	✗	Resource, location, and SLO awareness	OpenWhisk and custom
Hetzel et al.[32]	★★☆	N/A	★★☆	☆☆☆	✓	Resource awareness	Custom
Huang et al.[51]	★★★	N/A	★★★	★☆☆	✗	Resource awareness	Custom
Li et al.[46]	★★☆	YAML	☆☆☆	★★☆	✓	Resource and SLO awareness	Custom
Li et al.[47]	☆☆☆	N/A	☆☆☆	★★☆	✓	Resource and SLO awareness	Kubernetes and OpenFaaS
Baresi et al.[36]	☆☆☆	N/A	★★★	★★☆	✓	Resource, location, and SLO awareness	Kubernetes

Werner and Schirmer[18]	☆☆☆	N/A	☆☆☆	★★★	✗	Resource and SLO awareness	Custom
Serverless platform extensions							
Baresi and Quattrocchi[13]	☆☆☆	API	★☆☆	★☆☆	✓	Resource and SLO awareness	OpenFaaS
Benedetti et al.[19]	☆☆☆	YAML	★☆☆	★☆☆	✓	SLO awareness	OpenFaaS
Naranjo et al.[20]	★☆☆	N/A	★☆☆	★★☆	✗	Resource and SLO awareness	OSCAR[55]
Pérez et al.[55]	★☆☆	N/A	★☆☆	★★☆	✓	Resource awareness	OpenFaaS
Bermbach et al.[44]	☆☆☆	API	★★★	★☆☆	✓	Resource and SLO awareness	OpenWhisk
Wang et al.[33]	☆☆☆	API	★★☆	★★☆	✓	Resource and SLO awareness	OpenWhisk
Rausch et al.[31]	★★☆	N/A	★★☆	★★★	✓	Resource and location awareness	Kubernetes
Li et al.[56]	☆☆☆	API	☆☆☆	★☆☆	✗	Resource and SLO awareness	Knative
Das et al.[50]	☆☆☆	API and UI	★★☆	★★☆	✗	Resource and location awareness	AWS Lambda and Greengrass
Serverless programming extensions							
Zappa[21,‡]	★☆☆	N/A	☆☆☆	★☆☆	✓	Resource awareness	AWS Lambda
Chalice[22,‡]	★☆☆	N/A	☆☆☆	★☆☆	✓	Resource awareness	AWS Lambda
Zhang et al.[24]	★★★	API	☆☆☆	★☆☆	✓	Resource awareness	AWS Lambda
Chopra et al.[57]	★★★	API	☆☆☆	★☆☆	✓	Resource awareness	AWS Lambda
Burckhardt et al.[23]	★★☆	API	☆☆☆	★☆☆	✓	Resource awareness	Azure Functions
Burckhardt et al.[39]	★★★	API	☆☆☆	★☆☆	✓	Resource awareness	Azure Functions
Zhang et al.[10]	★★★	API	☆☆☆	★☆☆	✓	Resource awareness	Agnostic
Sreekanti et al.[9]	★★★	N/A	☆☆☆	★☆☆	✗	Resource awareness	Agnostic

*AI: artificial intelligence; API: application programming interface; N/A: not applicable; OSS: open source software; S & D: application state and data management; SLA: service level agreement; SLO: service level objective; UI: user interface.
†Commercial offering.
‡Open source project.

TABLE 2. Runtime.*

Reference	V	IR	ST	IT	G	OT	M	PI	HW
Core serverless platforms									
AWS Lambda[15,58,†]	MicroVM	★★★	☆☆☆	★★★	Centralized	N/A	N/A	N/A	✗
AWS Greengrass[40,†]	MicroVM	★★☆	★★☆	★★☆	Hybrid	N/A	N/A	N/A	✓
Google Cloud Functions[48,†]	Container	★★☆	☆☆☆	★★★	Centralized	N/A	N/A	N/A	✗
Microsoft Azure Functions[49,†]	Container	★★★	★★☆	★★★	Hybrid	N/A	N/A	N/A	✗
IBM Cloud Functions[53,†]	Container	★★☆	☆☆☆	★★★	Hybrid	N/A	N/A	N/A	✗
Apache OpenWhisk[43,‡]	Container	★★☆	☆☆☆	★★☆	Centralized	N/A	N/A	N/A	✓
OpenFaaS[16,‡]	Container	★★☆	★☆☆	★★☆	Centralized	Performance and resources	S	Analytical	✓
Knative[28,‡]	Container	★★☆	★☆☆	★★★	Centralized	Performance and resources	S	Analytical	✗
KServe[27,‡]	Container	★★☆	★☆☆	★★★	Centralized	Performance and resources	S	Analytical	✓
Nuclio[29,‡]	Container	★★☆	★☆☆	★★☆	Centralized	Resources	S	Analytical	✓
Fission[54,‡]	Container	☆	★☆☆	★★☆	Centralized	Performance and resources	S	Analytical	✗
Lordan et al.[34]	Container	★☆☆	☆☆☆	★★☆	Hybrid	✗	✗	✗	✓
Wolski et al.[45]	Container	★★★	★★★	★☆☆	Hybrid	Performance	R and S	✗	✗
Smith et al.[42]	Agnostic	★☆★	★☆☆	★☆☆	Centralized	Resources	P and S	✗	✗
Pfandzelter and Bermbach[17]	Container	★☆☆	★☆☆	★☆☆	Centralized	✗	R	✗	✗
Akkus et al.[52]	Container	★★☆	☆☆☆	★☆☆	Centralized	✗	✗	✗	✗
Hu et al.[35]	Container	★★☆	☆☆☆	★★☆	Distributed	Performance	S and R	Analytical	✓
Hetzel et al.[32]	MicroVM	★☆☆	★★☆	☆☆☆	Hybrid	✗	✗	✗	✗
Huang et al.[51]	Container	★☆☆	★★★	☆☆☆	Hybrid	Resources	P and S	ILP	✗
Li et al.[46]	Container	★☆☆	★☆☆	★☆☆	Centralized	Performance and resources	P	Heuristic	✗
Li et al.[47]	Container	★☆☆	★☆☆	★☆☆	Centralized	Cost and resources	S	Analytical	✗

Baresi et al.[36]	Container	★☆☆	★☆☆	★☆☆	Hybrid	Network and performance	R, P, and S	MIP	✓
Werner and Schirmer[18]	Native	★☆☆	★☆☆	★☆☆	Centralized	✗	✗	✗	✓
Serverless platform extensions									
Li et al.[46]	Container	★☆☆	★★☆	☆☆☆	Centralized	Resources	P and S	Heuristic	✗
Baresi et al.[13]	Container	★★☆	★☆☆	★★☆	Hybrid	Resources and performance	R, P, and S	MIP and heuristic	✗
Benedetti and Quattrocchi[19]	Container	★★☆	★☆☆	★★☆	Centralized	Resources	S	RL	✗
Naranjo et al.[20]	Container	★★☆	★☆☆	★☆☆	Centralized	Performance	P	✗	✓
Pérez et al.[55]	Container	★★☆	★☆☆	★☆☆	Centralized	CLUES[59]	R, P, and S	✗	✗
Bermbach et al.[44]	Container	★★☆	☆☆☆	★★☆	Distributed	Revenue	P and R	Auction	✗
Wang et al.[33]	Container	★★☆	☆☆☆	★★★	Centralized	Performance	R, P, and S	Model driven	✗
Rausch et al.[31]	Container	★☆☆	★☆☆	★☆☆	Centralized	Performance and network	P	✗	✓
Bendetti et al.[19]	Container	★★☆	★☆☆	★★☆	Centralized	Resources	S	RL	✗
Li et al.[56]	Container	★★☆	★☆☆	★★★	Centralized	Performance	P and S	Analytical	✗
Das et al.[50]	MicroVM	★★★	★★☆	★★★	Hybrid	Cost and performance	P	ML and analytical	✗
Serverless function runtimes									
Cloudflare Workers[60,†]	V8	★★☆	★★★	★★☆	Hybrid	✗	✗	✗	✗
Shillaker and Pietzuch[26]	WASM	★★☆	★★★	★★★	Centralized	Performance and resources	S	Analytical	✗
Gadepalli et al.[61]	WASM	N/A	☆☆☆	★☆☆	N/A	Performance and resources	R and S	Work stealing	✗
Oakes et al.[62]	Container	N/A	☆☆☆	★☆☆	N/A	✗	✗	✗	✗
Karhula et al. [38]	Container	N/A	★★★	N/A	N/A	N/A	N/A	N/A	N/A
Gackstatter et al.[25]	WASM	★★☆	☆☆☆	★★☆	Centralized	✗	✗	✗	✗

*G: edge–cloud gateways; HW: hardware accelerator; IR: invocation reliability; IT: invocation triggers; M: runtime mechanisms (i.e., scaling, placement, and routing); ML: machine learning; N/A: not applicable; OT: performance and resource optimization targets; PI: platform intelligence; ST: stateful function support; V: virtualization.
†Commercial offering.
‡Open source project.

for customers using *analytical* and *ML-driven platform* intelligence,[47,50] or increase revenue using auctions (i.e., *optimization-based platform intelligence*).[44] Besides focusing on *performance and cost*, our review also includes approaches that focus on *resources* and *network*. Specifically, Huang et al.[51] use integer linear programming to enable function migrations to efficiently use *compute resources*; other approaches reduce *network* traffic by moving function instances toward the data[31] or reduce network latency by dynamically creating computing clusters with nodes close to each other and network-aware routing.[36]

Edge Support

Commercial platforms offer two types of edge-oriented extensions: custom edge platforms[40] that integrate with cloud platforms and function executions at predefined CDN nodes.[2] The former allows the inclusion of custom devices but entails custom management to combine cloud and edge resources.[50] The latter enables developers to pre- and postprocess HTTP requests and responses.[2] Open source platforms rarely support edge devices and focus on providing resource-optimized platforms (i.e., OpenWhisk Lean[43]). To summarize, commercial and open source platforms offer high *edge support* maturity, but their solutions lack the unification of edge and cloud resources, as we envision the SCF.

Core serverless platforms and *serverless platform extensions* from academia offer high maturity and unify the edge and cloud. For example, Wolski et al.[45] implement a platform that runs on devices ranging from microcontrollers to consumer workstations using a distributed fault-tolerant, append-only storage to guarantee function executions across the continuum. On the contrary, Akkus et al.[52] introduce a local message bus deployed per node, making the approach resilient against network failures, as functions on the same node can still communicate, and their resource consumption results show a smaller footprint than AWS Greengrass. Other approaches achieve a high maturity level of *edge support* by enabling the seamless migration of function instances in edge–cloud scenarios[51] or by introducing data-aware request routing and data replication that gradually lead to reduced network traffic between regions.[42]

WHAT LIES AHEAD

Programming Support

Our review has shown that only a few approaches offer mature *AI support* and *application and state data management*. Specifically, we see a lack of programming support for first-class citizens abstractions.

EI: Developing EI applications still remains challenging because of their complex *workflows* that combine edge and cloud resources to perform training and inference.[63] We saw that current platforms do not adequately address this, and new solutions are required. For example, AI model selection and the treatment of large AI models should be first-class-citizen features in the programming model.[18]

Application state and data management still poses great challenges due to the edge–cloud continuum's heterogeneous network conditions and geodistributed infrastructure. Thus, offering distributed cache and storage solutions is not only mandatory but also challenging in terms of *implementation* and usage. For example, the programming model should assist developers by offering appropriate abstractions that simplify the access.

Reliability Engineering

The review has shown that only a few approaches have reached full maturity in terms of guaranteeing efficient execution. Specifically, support for SLOs and execution guarantees still poses some challenges.

SLO/SLA awareness remains challenging in terms of enabling developers to specify complex SLOs.[64,65] For example, developers want to specify that the inference model has a certain accuracy.[63] Furthermore, the implementation requires careful design to monitor and manage function executions efficiently. For example, the SCF should continuously learn and build AI models to make informed function scaling, placement, and routing decisions.

Reliable invocations pose challenges in the implementation because each *edge–cloud gateway* approach has its advantages and disadvantages. For example, the *hybrid* approach can introduce additional network overhead by routing through gateway components, while a drawback of the *distributed* approach is the dissemination of knowledge in the system. Further, execution guarantees for a mission-critical complex are hard to achieve in such dynamic environments. For example, the SCF can deploy a distributed queuing mechanism that can run on all devices to achieve full maturity.

Performance and Infrastructure Engineering

Current approaches lack sophisticated *optimization* strategies. Specifically, commercial and community-driven open source solutions do not implement adequate ones that are suitable for the edge–cloud continuum. In addition, the maturity of *edge support* is high in approaches from academia but lacks adoption in the open source space.

Optimization based on *ML/AI-driven platform intelligence* can continuously adapt to the dynamic edge–cloud environment and make smart scaling and placement decisions. However, we identify two challenging tasks: 1) placement and 2) operationalization. 1) The SCF must consider the implementation and placement of runtime mechanism components (e.g., scaling and placement). Each of those components can be deployed distributed and decentralized, requiring coordination and fallback mechanisms. 2) The operationalization of intelligent placement, scaling, and routing is complex and requires retraining due to the dynamic environment of the edge–cloud continuum.

Edge support is challenging to offer due to the large variety of devices and range of computational power. To improve this, we suggest that the SCF should abstract the virtualization and isolation layer to enable a resource-efficient and resilient function execution model on all devices and reach full maturity. Containers, WebAssembly, and microVMs each have advantages and disadvantages,[25,66,67] and developers should have the option to let the SCF choose the most suitable one dynamically.

RELATED WORK

Aslanpour et al.[2] identify opportunities and open issues. Certain aspects of our work overlap with theirs, but we additionally assess the current state. Cassel et al.[68] and Kjorveziroski et al.[69] review serverless computing for the IoT. While the former do not investigate commercial or open source platforms, the latter neglect stateful aspects. Ioini et al.[70] focus on the state of commercial and open source platforms. Our article generally focuses on different issues and challenges and presents a novel look at serverless edge computing with our criteria that investigate the programming and execution model.

CONCLUSION

In this article, we introduced novel criteria to evaluate the maturity of serverless computing approaches for the edge–cloud continuum. Our work is, in part, a continuation of previous research where we outlined our vision for the SCF.[4] Our review of more than 45 approaches discusses challenges we previously identified and evaluates the maturity of current offerings. We identified issues around data management, the serverless programming model, and AI support. The review shows that current offerings are, in many ways, still immature, and we have briefly outlined which steps to take next.

REFERENCES

1. S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar, and A. Y. Zomaya, "Edge intelligence: The confluence of edge computing and artificial intelligence," *IEEE Internet Things J.*, vol. 7, no. 8, pp. 7457–7469, Aug. 2020, doi: 10.1109/JIOT.2020.2984887.
2. M. S. Aslanpour et al., "Serverless edge computing: Vision and challenges," in *Proc. Australas. Comput. Sci. Week Multiconference*, Feb. 2021, pp. 1–10, doi: 10.1145/3437378.3444367.
3. H. Shafiei, A. Khonsari, and P. Mousavi, "Serverless computing: A survey of opportunities, challenges, and applications," *ACM Comput. Surv.*, vol. 54, no. 11s, pp. 1–32, Nov. 2022, doi: 10.1145/3510611.
4. S. Nastic, P. Raith, A. Furutanpey, T. Pusztai, and S. Dustdar, "A serverless computing fabric for edge and cloud," in *Proc. IEEE 4th Int. Conf. Cogn. Mach. Intell. (CogMI)*, 2022, pp. 1–12, doi: 10.1109/CogMI56440.2022.00011.
5. S. Nastic et al., "Polaris scheduler: Edge sensitive and SLO aware workload scheduling in cloud-edge-IoT clusters," in *Proc. IEEE 14th Int. Conf. Cloud Comput. (CLOUD)*, 2021, pp. 206–216, doi: 10.1109/CLOUD53861.2021.00034.
6. T. Pusztai et al., "Polaris scheduler: SLO- and topology-aware microservices scheduling at the edge," in *Proc. IEEE/ACM 15th Int. Conf. Utility Cloud Comput. (UCC)*, 2022, pp. 61–70, doi: 10.1109/UCC56403.2022.00017.
7. P. Raith, T. Rausch, S. Dustdar, F. Rossi, V. Cardellini, and R. Ranjan, "Mobility-aware serverless function adaptations across the edge-cloud continuum," in *Proc. IEEE/ACM 15th Int. Conf. Utility Cloud Comput. (UCC)*, 2022, pp. 123–132, doi: 10.1109/UCC56403.2022.00023.
8. V. Prokhorenko and M. A. Babar, "Architectural resilience in cloud, fog and edge systems: A survey," *IEEE Access*, vol. 8, pp. 28,078–28,095, Feb. 2020, doi: 10.1109/ACCESS.2020.2971007.
9. V. Sreekanti, C. Wu, S. Chhatrapati, J. E. Gonzalez, J. M. Hellerstein, and J. M. Faleiro, "A fault-tolerance shim for serverless computing," in *Proc. 15th Eur. Conf. Comput. Syst.*, 2020, pp. 1–15, doi: 10.1145/3342195.3387535.
10. H. Zhang, A. Cardoza, P. B. Chen, S. Angel, and V. Liu, "Fault-tolerant and transactional stateful serverless workflows," in *Proc. 14th USENIX Symp. Operating Syst. Des. Implementation (OSDI)*, 2020, pp. 1187–1204.
11. A. Jangda, D. Pinckney, Y. Brun, and A. Guha, "Formal foundations of serverless computing," *Proc. ACM Program. Lang.*, vol. 3, no. OOPSLA, pp. 1–26, Oct. 2019, doi: 10.1145/3360575.

12. J. Scheuner et al., "TriggerBench: A performance benchmark for serverless function triggers (short paper)," in *Proc. IEEE Int. Conf. Cloud Eng. (IC2E)*, 2022, pp. 96–103, doi: 10.1109/IC2E55432.2022.00018.
13. L. Baresi and G. Quattrocchi, "PAPS: A serverless platform for edge computing infrastructures," *Frontiers Sustain. Cities*, vol. 3, Jul. 2021, Art. no. 690660, doi: 10.3389/frsc.2021.690660.
14. "Kubeless." GitHub. Accessed: Mar. 29, 2023. [Online]. Available: https://github.com/vmware-archive/kubeless
15. "AWS Lambda." Amazon. Accessed: Mar. 29, 2023. [Online]. Available: https://aws.amazon.com/lambda/
16. OpenFaaS. [Online]. Available: https://www.openfaas.com/
17. T. Pfandzelter and D. Bermbach, "tinyFaaS: A lightweight FaaS platform for edge environments," in *Proc. IEEE Int. Conf. Fog Comput. (ICFC)*, 2020, pp. 17–24, doi: 10.1109/ICFC49376.2020.00011.
18. S. Werner and T. Schirmer, "HARDLESS: A generalized serverless compute architecture for hardware processing accelerators," in *Proc. IEEE Int. Conf. Cloud Eng. (IC2E)*, 2022, pp. 79–84, doi: 10.1109/IC2E55432.2022.00016.
19. P. Benedetti, M. Femminella, G. Reali, and K. Steenhaut, "Reinforcement learning applicability for resource-based auto-scaling in serverless edge applications," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops Other Affiliated Events (PerCom Workshops)*, 2022, pp. 674–679, doi: 10.1109/PerComWorkshops53856.2022.9767437.
20. D. M. Naranjo, S. Risco, C. de Alfonso, A. Pérez, I. Blanquer, and G. Moltó, "Accelerated serverless computing based on GPU virtualization," *J. Parallel Distrib. Comput.*, vol. 139, pp. 32–42, May 2020, doi: 10.1016/j.jpdc.2020.01.004.
21. "Zappa." GitHub. Accessed: Mar. 29, 2023. [Online]. Available: https://github.com/zappa/Zappa
22. "Chalice." GitHub. Accessed: Mar. 29, 2023. [Online]. Available: https://github.com/aws/chalice
23. S. Burckhardt, C. Gillum, D. Justo, K. Kallas, C. McMahon, and C. S. Meiklejohn, "Durable functions: Semantics for stateful serverless," *Proc. ACM Program. Lang.*, vol. 5, no. OOPSLA, pp. 1–27, Oct. 2021, doi: 10.1145/3485510.
24. W. Zhang, V. Fang, A. Panda, and S. Shenker, "Kappa: A programming framework for serverless computing," in *Proc. 11th ACM Symp. Cloud Comput.*, 2020, pp. 328–343, doi: 10.1145/3419111.3421277.
25. P. Gackstatter, P. A. Frangoudis, and S. Dustdar, "Pushing serverless to the edge with webassembly runtimes," in *Proc. 22nd IEEE Int. Symp. Cluster, Cloud Internet Comput. (CCGrid)*, 2022, pp. 140–149, doi: 10.1109/CCGrid54584.2022.00023.
26. S. Shillaker and P. Pietzuch, "FAASM: Lightweight isolation for efficient stateful serverless computing," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2020, pp. 419–433.
27. "Master." Kserve. Accessed: Mar. 29, 2023. [Online]. Available: https://kserve.github.io/website/master/
28. Knative. Accessed: Mar. 29, 2023. [Online]. Available: https://knative.dev/docs/
29. Nuclio. Accessed: Mar. 29, 2023. [Online]. Available: https://nuclio.io/
30. J. Li, S. G. Kulkarni, K. Ramakrishnan, and D. Li, "Understanding open source serverless platforms: Design considerations and performance," in *Proc. 5th Int. Workshop Serverless Comput.*, 2019, pp. 37–42, doi: 10.1145/3366623.3368139.
31. T. Rausch, A. Rashed, and S. Dustdar, "Optimized container scheduling for data-intensive serverless edge computing," *Future Gener. Comput. Syst.*, vol. 114, pp. 259–271, Jan. 2021, doi: 10.1016/j.future.2020.07.017.
32. R. Hetzel, T. Kärkkäinen, and J. Ott, "μactor: Stateful serverless at the edge," in *Proc. 1st Workshop Serverless Mobile Netw. 6G Commun.*, 2021, pp. 1–6, doi: 10.1145/3469263.3470828.
33. B. Wang, A. Ali-Eldin, and P. Shenoy, "Lass: Running latency sensitive serverless computations at the edge," in *Proc. 30th Int. Symp. High-Perform. Parallel Distrib. Comput.*, 2021, pp. 239–251, doi: 10.1145/3431379.3460646.
34. F. Lordan, D. Lezzi, and R. M. Badia, "Colony: Parallel functions as a service on the cloud-edge continuum," in *Proc. Eur. Conf. Parallel Process.*, Cham, Switzerland: Springer-Verlag, 2021, pp. 269–284, doi: 10.1007/978-3-030-85665-6_17.
35. J. Hu et al., "HiveMind: A scalable and serverless coordination control platform for UAV swarms," 2020, *arXiv:2002.01419*.
36. L. Baresi, D. Y. X. Hu, G. Quattrocchi, and L. Terracciano, "NEPTUNE: Network- and GPU-aware management of serverless functions at the edge," in *Proc. 17th Symp. Softw. Eng. Adaptive Self-Manag. Syst. (SEAMS)*, New York, NY, USA: Association for Computing Machinery, 2022, pp. 144–155, doi: 10.1145/3524844.3528051.
37. Z. Li, L. Guo, J. Cheng, Q. Chen, B. He, and M. Guo, "The serverless computing survey: A technical primer for design architecture," *ACM Comput. Surv.*, vol. 54, no. 10s, pp. 1–34, doi: 10.1145/3508360.
38. P. Karhula, J. Janak, and H. Schulzrinne, "Checkpointing and migration of IoT edge functions,"

in *Proc. 2nd Int. Workshop Edge Syst., Analytics Netw.*, 2019, pp. 60–65, doi: 10.1145/3301418.3313947.
39. S. Burckhardt et al., "Netherite: Efficient execution of serverless workflows," *Proc. VLDB Endowment*, vol. 15, no. 8, pp. 1591–1604, Apr. 2022, doi: 10.14778/3529337.3529344.
40. "AWS Greengrass." Amazon. [Online]. Available: https://docs.aws.amazon.com/greengrass/index.html#aws-iot-greengrass,-version-2
41. R. Klingler, N. Trifunovic, and J. Spillner, "Beyond@cloudfunction: Powerful code annotations to capture serverless runtime patterns," in *Proc. 7th Int. Workshop Serverless Comput. (WoSC)*, 2021, pp. 23–28, doi: 10.1145/3493651.3493669.
42. C. P. Smith, A. Jindal, M. Chadha, M. Gerndt, and S. Benedict, "FaDO: FaaS functions and data orchestrator for multiple serverless edge-cloud clusters," in *Proc. IEEE 6th Int. Conf. Fog Edge Comput. (ICFEC)*, 2022, pp. 17–25, doi: 10.1109/ICFEC54809.2022.00010.
43. "IBM." Apache OpenWhisk. Accessed: Mar. 29, 2023. [Online]. Available: https://openwhisk.apache.org/
44. D. Bermbach, J. Bader, J. Hasenburg, T. Pfandzelter, and L. Thamsen, "Auctionwhisk: Using an auction-inspired approach for function placement in serverless fog platforms," *Softw. Pract. Experience*, vol. 52, no. 5, pp. 1143–1169, May 2022, doi: 10.1002/spe.3058.
45. R. Wolski, C. Krintz, F. Bakir, G. George, and W.-T. Lin, "Cspot: Portable, multi-scale functions-as-a-service for IoT," in *Proc. 4th ACM/IEEE Symp. Edge Comput.*, 2019, pp. 236–249, doi: 10.1145/3318216.3363314.
46. Z. Li et al., "FaaSFlow: Enable efficient workflow execution for function-as-a-service," in *Proc. 27th ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2022, pp. 782–796, doi: 10.1145/3503222.3507717.
47. X. Li, P. Kang, J. Molone, W. Wang, and P. Lama, "KneeScale: Efficient resource scaling for serverless computing at the edge," in *Proc. 22nd IEEE Int. Symp. Cluster, Cloud Internet Comput. (CCGrid)*, 2022, pp. 180–189, doi: 10.1109/CCGrid54584.2022.00027.
48. "Cloud functions." Google. Accessed: Mar. 29, 2023. [Online]. Available: https://cloud.google.com/functions
49. "Azure Functions documentation." Microsoft. Accessed: Mar. 29, 2023. [Online]. Available: https://docs.microsoft.com/en-us/azure/azure-functions/
50. A. Das, S. Imai, S. Patterson, and M. P. Wittie, "Performance optimization for edge-cloud serverless platforms via dynamic task placement," in *Proc. 20th IEEE/ACM Int. Symp. Cluster, Cloud Internet Comput. (CCGRID)*, 2020, pp. 41–50, doi: 10.1109/CCGrid49817.2020.00-89.
51. Y. Huang, Z. Lin, T. Yao, X. Shang, L. Cui, and J. Z. Huang, "Mobility-aware seamless virtual function migration in deviceless edge computing environments," in *Proc. IEEE 42nd Int. Conf. Distrib. Comput. Syst. (ICDCS)*, 2022, pp. 447–457, doi: 10.1109/ICDCS54860.2022.00050.
52. I. E. Akkus et al., "SAND: Towards High-Performance serverless computing," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2018, pp. 923–935.
53. "IBM cloud functions." IBM. [Online]. Available: https://cloud.ibm.com/functions/
54. Fission. Accessed: Mar. 29, 2023. [Online]. Available: https://fission.io/
55. A. Pérez, S. Risco, D. M. Naranjo, M. Caballer, and G. Moltó, "On-premises serverless computing for event-driven data processing applications," in *Proc. IEEE 12th Int. Conf. Cloud Comput. (CLOUD)*, 2019, pp. 414–421, doi: 10.1109/CLOUD.2019.00073.
56. D. Li et al., "SoDa: A serverless-oriented deadline-aware workflow scheduling engine for IoT applications in edge clouds," *Wireless Commun. Mobile Comput.*, vol. 2022, Oct. 2022, Art. no. 7862911, doi: 10.1155/2022/7862911.
57. A. K. Chopra et al., "Deserv: Decentralized serverless computing," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, 2021, pp. 51–60, doi: 10.1109/ICWS53863.2021.00020.
58. A. Agache et al., "Fire-cracker: Lightweight virtualization for serverless applications," in *Proc. 17th USENIX Symp. Networked Syst. Des. Implementation (NSDI)*, 2020, pp. 419–434.
59. C. de Alfonso, M. Caballer, A. Calatrava, G. Moltó, and I. Blanquer, "Multi-elastic datacenters: Auto-scaled virtual clusters on energy-aware physical infrastructures," *J. Grid Comput.*, vol. 17, no. 1, pp. 191–204, Mar. 2019, doi: 10.1007/s10723-018-9449-z.
60. "How workers works." Cloudflare. Accessed: Mar. 29, 2023. [Online]. Available: https://developers.cloudflare.com/workers/learning/how-workers-works/
61. P. K. Gadepalli, S. McBride, G. Peach, L. Cherkasova, and G. Parmer, "Sledge: A serverless-first, light-weight wasm runtime for the edge," in *Proc. 21st Int. Middleware Conf.*, Dec. 2020, pp. 265–279, doi: 10.1145/3423211.3425680.
62. E. Oakes et al., "SOCK: Rapid task provisioning with serverless-optimized containers," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2018, pp. 57–70.
63. P. Raith and S. Dustdar, "Edge intelligence as a service," in *Proc. IEEE Int. Conf. Services Comput. (SCC)*, 2021, pp. 252–262.

64. T. Pusztai et al., "SLO script: A novel language for implementing complex cloud-native elasticity-driven SLOs," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, 2021, pp. 21–31, doi: 10.1109/ICWS53863.2021.00017.
65. T. Pusztai et al., "A novel middleware for efficiently implementing complex cloud-native SLOs," in *Proc. IEEE 14th Int. Conf. Cloud Comput. (CLOUD)*, 2021, pp. 410–420, doi: 10.1109/CLOUD53861.2021.00055.
66. Z. Li et al., "Help rather than recycle: Alleviating cold startup in serverless computing through Inter-Function container sharing," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2022, pp. 69–84.
67. T. Goethals, M. Sebrechts, M. Al-Naday, B. Volckaert, and F. De Turck, "A functional and performance benchmark of lightweight virtualization platforms for edge computing," in *Proc. IEEE Int. Conf. Edge Comput. Commun. (EDGE)*, 2022, pp. 60–68, doi: 10.1109/EDGE55608.2022.00020.
68. G. A. S. Cassel, V. F. Rodrigues, R. da Rosa Righi, M. R. Bez, A. C. Nepomuceno, and C. A. da Costa, "Serverless computing for internet of things: A systematic literature review," *Future Gener. Comput. Syst.*, vol. 128, pp. 299–316, Mar. 2022, doi: 10.1016/j.future.2021.10.020.
69. V. Kjorveziroski, S. Filiposka, and V. Trajkovic, "IoT serverless computing at the edge: Open issues and research direction," *Computers*, vol. 10, no. 10, 2021, Art. no. 130, doi: 10.3390/computers10100130.
70. N. E. Ioini, D. Hästbacka, C. Pahl, and D. Taibi, "Platforms for serverless at the edge: A review," in *Proc. Eur. Conf. Service-Oriented Cloud Comput.*, Cham, Switzerland: Springer-Verlag, 2020, pp. 29–40, doi: 10.1007/978-3-030-71906-7_3.

PHILIPP RAITH is Ph.D. student and university assistant at the Distributed Systems Group, TU Wien, Wein, 1040, Austria. His research interests include serverless edge computing, resource management in the edge–cloud continuum, and edge intelligence. Raith received his M.Sc. in computer science from TU Wien. Contact him at p.raith@dsg.tuwien.ac.at.

STEFAN NASTIC is an assistant professor on a tenure track at Distributed Systems Group, TU Wien, Wein, 1040, Austria, and is also a founder and managing director of IntelliEdge GmbH. His research interests include serverless computing, the edge–cloud continuum, artificial intelligence (AI) and edge AI, and reliability engineering. Nastic received his Ph.D. in programming, provisioning, and governing IoT cloud systems from TU Wien. Contact him at snas-tic@dsg.tuwien.ac.at.

SCHAHRAM DUSTDAR is a full professor of computer science heading the Distributed Systems Group, TU Wien, Wein, 1040, Austria. His research focuses on distributed systems. He is a Fellow of IEEE. Contact him at dustdar@dsg.tuwien.ac.at.