A Serverless Computing Fabric for Edge & Cloud

Stefan Nastic, Philipp Raith, Alireza Furutanpey, Thomas Pusztai, Schahram Dustdar
Distributed Systems Group,
TU Wien, Vienna, Austria

***Abstract*—Serverless computing has been establishing itself as a compelling paradigm for the development and of modern cloud-native applications. Serverless represents the next step in the evolution of cloud programming models, services and platforms, which is especially appealing due to its low management overhead, easy deployment, scale-to-zero and the promise of optimized costs. Recently, due to the advantages it offers, the serverless paradigm has been growing beyond traditional clouds, making its way to the Edge. The natural evolutionary step for serverless computing is to unify the Edge and the Cloud into what we refer to as Edge-Cloud Continuum. In this paper, we outline our vision of the Serverless Computing Fabric (SCF) for the Edge-Cloud continuum. We introduce the reference architecture for the SCF and show how it unlocks the full potential of the Edge-Cloud continuum. We also discuss main opportunities and challenges, which need to be overcome in order to achieve the vision of the Serverless Computing Fabric. Finally, we introduce key design principles together with core enabling runtime mechanisms, which are intended to serve as a research road map towards the Serverless Computing Fabric for Edge-Cloud continuum.**

***Index Terms*—Serverless Computing, Cloud Computing, Edge Computing, Reliability Engineering, Service Level Objectives**

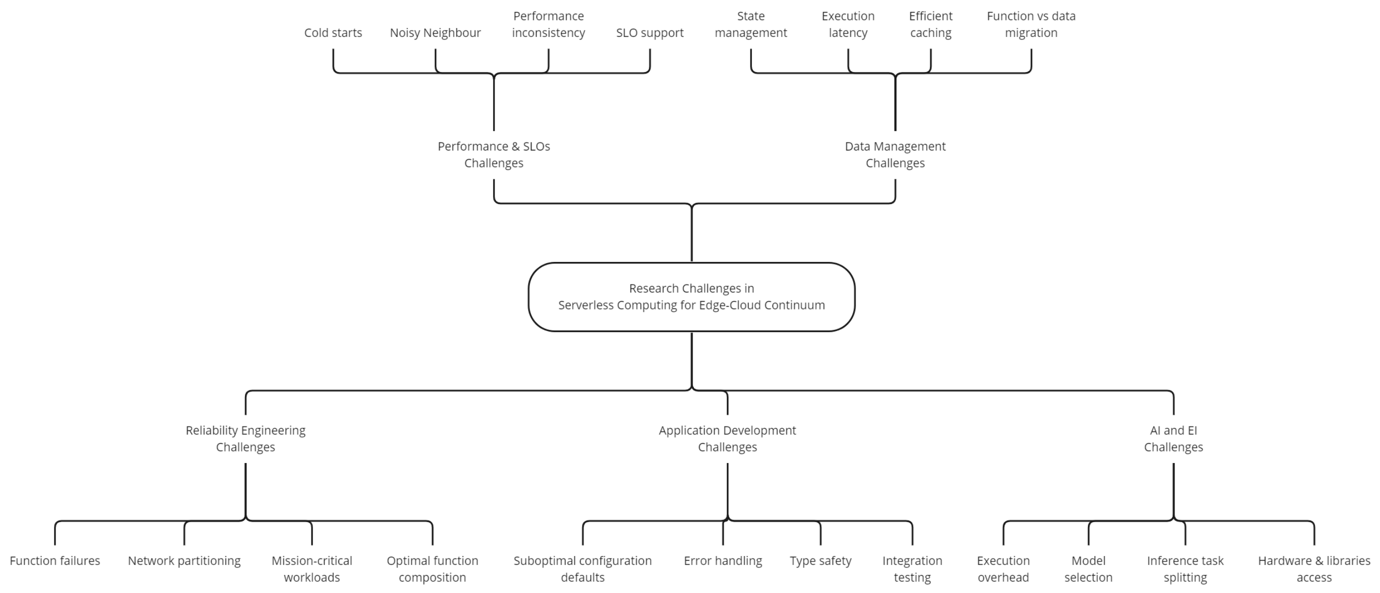
I. Introduction

With the increasing growth of edge computing, fog computing and the Internet of Things (IoT), abundant computing infrastructure and edge resources are becoming available and increasingly utilized by various applications. Currently there are many definitions of edge computing. Some researchers view the Edge as an extension of cloud's content delivery networks [36], while others propose more edge-enteric view [15]. We believe that most of such exclusionary views on edge computing fail to capture its full potential. To fully tap into the potential of edge computing, we believe that a more holistic view, which considers both Edge and Cloud is better suited. In particular, we view Edge as a main constituent of a large-scale, geographically distributed and hierarchically organized compute continuum. We refer to it as Edge-Cloud continuum. The Edge-Cloud continuum spans from the centralized cloud to the far edge of the infrastructure to offer computing, networking and data resources. At the same time, the Edge-Cloud continuum poses new challenges (e.g., [41]) and calls for new approaches to enable delivering such resources in a uniform manner.

Serverless computing is an emerging paradigm, which typically refers to a software architecture where an application is decomposed into "trigger" and "actions" or functions. Typically there is a platform, which enables seamless hosting and execution of such developer-defined functions, making it easy to develop, manage, scale, and operate those functions. The complexity mitigation is achieved by incorporating sophisticated runtime mechanisms into the serverless platform and relieving the users from those responsibilities. Function as a Service or FaaS is the underlying programming and execution model for Serverless computing. FaaS model is the core of the serverless computing and sometimes it is even equated with serverless computing. However, without suitable supporting services, e.g., for persistent storage, and native interfaces, e.g., to access network or file system, FaaS on its own has limited practical benefits. Because of this, Serverless has been evolving into an ecosystem, typically comprising a FaaS platform and a large number of supporting cloud services [21]. Moreover, increasing number of traditional services such as DBaaS have been appearing in a serverless or more precisely a FaaS flavor. Therefore, serverless computing can be considered as the next step in the evolution of cloud computing or more generally of utility computing.

Although Serverless was born in the cloud, its true potential is unlocked at the Edge, where the lightweight and dynamic FaaS functions can take full advantage of the vast and geo-distributed infrastructure. Conversely, Serverless has the potential to enable and foster proliferation of Edge computing, by reducing the complexity that is currently associated with developing and operating Edge applications. Additionally, main serverless properties such as no idle execution and out-of-the-box scalability are particularly useful to bring much desired improvements to the reliability and performance of the Edge-Cloud applications. Finally, specific areas such as Edge Intelligence (EI) and AI can particularly benefit from enabling the serverless principles and models at the Edge, since EI and AI applications are notoriously hard to operate, but at the same time they significantly benefit from specialized infrastructure (e.g., AI inference accelerators). Unfortunately, still to date the serverless paradigm typically remains limited to the cloud.

In this paper, we propose a novel **Serverless Computing Fabric (SCF)** for the Edge-Cloud continuum. The SCF is a continuation of our previous work in Serverless and Deviceless Edge Computing [16], [26] and presents our view on the serverless paradigm in the Edge-Cloud continuum. We introduce a reference architecture for the SCF and show how it unlocks the full potential of the Edge-Cloud continuum. This is mainly achieved by abstracting away infrastructure complexities, offering reliable application execution, and offering adequate programming support. We particularly focus



Cold starts
Noisy Neighbour
Performance inconsistency
SLO support
State management
Execution latency
Efficient caching
Function vs data migration
Performance & SLOs Challenges
Data Management Challenges
Research Challenges in Serverless Computing for Edge-Cloud Continuum
Reliability Engineering Challenges
Application Development Challenges
AI and EI Challenges
Function failures
Network partitioning
Mission-critical workloads
Optimal function composition
Suboptimal configuration defaults
Error handling
Type safety
Integration testing
Execution overhead
Model selection
Inference task splitting
Hardware & libraries access

Fig. 1: Overview of Main Research Challenges in Serverless Computing in Edge-Cloud Continuum

on how the SCF fosters novel EI and AI applications. Finally, we introduce key design principles and core enabling runtime mechanisms, which are intended to serve as a research road map towards the Serverless Computing Fabric for Edge-Cloud continuum.

The remainder of the paper is structured as follows. In Section II, we discuss the main research challenges in the Serverless Computing Fabric. We also look at the most important opportunities unlocked by SCF in the Edge-Cloud continuum. Section III introduces our vision for the SCF and Section IV provides detailed descriptions of the main design principles and the core enabling mechanisms of the SCF. In Section V, we discuss the related work. Finally, Section VI concludes the paper.

II. Opportunities & Research Challenges in Serverless Computing for Edge-Cloud Continuum

The Edge-Cloud continuum resources such as compute and storage are poised to become utility, in the sense that Edge-Cloud continuum can deliver computing resources the way a power utility doles out electricity. Serverless is a perfectly suitable paradigm and execution model to enable this vision due to its promise of minimal operation overhead, transparent scaling, no idle execution (due to scale-to-zero feature) and turn-key high availability.

More specifically, benefits of combining Serverless and Edge paradigms, include:

- *Edge-native backend services* - Serverless Computing enables minimal operational management of fine-grained functions, but it also requires a multitude Backend-as-a-Service (BaaS) to be able to develop serverless applications that go beyond toy examples. Examples of such BaaS services include storage, messaging middleware, caching solutions, and so forth. BaaS not only facilitates FaaS in general, but it can also provide essential backend services for specialized applications and service paradigms. For example, streaming IoT platforms require reliable and robust messaging services to process data. Low latency applications benefit from caching and storage solutions close to the application instance. Moreover, in contrast to short-running functions, backend services are long-running and require special attention in resource-constrained environments to not put too much strain on devices. Storage solutions have to adapt and migrate due to the geo-distributed nature of the Edge-Cloud continuum and accommodate toward user mobility (i.e., cars). Further, messaging middleware must be placed toward the user to guarantee low latency and additionally reliable transmission. Reliability is a key concern in systems with unstable network connections. Caching solutions enable low latency applications but have to intelligently manage the cached content to not overload nodes. A dedicated set of runtime mechanisms to manage and provide reliable backend services enables this opportunity.
- *Sustainable service delivery at the Edge* - Traditional deployment models typically demand long-term running servers in order to enable sustainable service delivery. However, this exclusive allocation needs to retain resources regardless of whether the user application is running or not. A minimal resource footprint is crucial for Edge-Cloud systems as the infrastructure is heterogeneous, and devices toward the edge tend to be resource-constrained. Serverless Edge Computing can facilitate minimizing resource usage due to the elasticity of functions. FaaS platforms also allow complete control over deployments making it possible to apply resource

management strategies that optimize toward resource efficiency and can reduce carbon emissions. Additionally, they can scale down the number of function instances to zero if they are not needed. The combination of an extended serverless programming model, an elastic system to monitor and enforce SLOs, and an intelligent and autonomous orchestration will enable sustainable service delivery across the Edge-Cloud Continuum.

- *Enabling Edge Intelligence* - EI is characterised by an emphasis on AI-based applications that flourish in the Edge-Cloud continuum [12]. These applications range from short-running inference tasks (e.g., Mobile Augmented Reality) to long-running and resource-demanding model training jobs. Besides resource and performance requirements, EI applications are heterogeneous. They have a unique position across applications as the code itself might be simple, but the used models differ vastly in execution. Emerging approaches to split models and perform inference across multiple devices [24], as well as distributed training of models [20], further increase the complexity of managing these deployments. Specifically, Serverless Edge Computing enables platform providers to deploy strategies that can take the various characteristics into account that make EI applications complex to manage. This includes an intelligent control plane and an elastic data plane. The control plane includes pro-active scaling approaches, intelligent and distributed high-throughput schedulers, and location- and network-aware request routing approaches. The elastic data plane must adapt to the dynamic nature of Edge-Cloud infrastructures. This dynamic part includes nodes that leave and re-join, unstable network connections and heterogeneous devices with different capabilities. Current approaches [33], [44] explore already the potential of Serverless Edge Computing platforms. Still, we show in this paper our vision of how to go further to enable Edge Intelligence using Serverless Edge Computing.

A. Main Research Challenges

1) Performance and SLO Challenges: Despite flexible scaling and high availability offered by the serverless paradigm, there are still a number of performance and SLO challenges, which need to be dealt with when operating serverless platforms and managing serverless applications. We identify the most important challenges, which can affect the serverless workloads in terms of performance degradation and SLO violations.

- Startup and scheduling latency (a.k.a “cold start”).
- Lack of performance isolation (a.k.a “noisy neighbor” or performance interference [46]).
- Inconsistent performance due to hardware heterogeneity (also present in Cloud only solutions!).
- Limited support for application SLOs. Typically, one can only specify memory requirements, function timeouts and concurrency.

2) Data Management Challenges: Serverless FaaS platforms typically provide programming and execution support to the serverless functions, hence they are almost exclusively focused on managing the compute resources. Managing data and function states largely remains user’s concern. The most important challenges related to data management in serverless applications include:

- State management for stateful computations.
- Function execution latency due to lack of data locality.
- Efficient and cost-effective caching solutions.
- Trade-off between data and function execution movement (i.e., move execution where data sits or move data to "fast" hardware)

3) Reliability Engineering Challenges: Reliability engineering encompasses the ability of a workload to perform its intended function correctly and consistently at any given time, i.e., when it is expected to. Except for the reliability guarantees for side-effect-free stateless computations, there are numerous reliability engineering challenges including:

- Dealing with function failures beyond simple retries [19].
- Mitigating network partitioning, which can render functions useless (e.g., due to detached storage).
- Operating fault-tolerant mission-critical Edge workloads.
- Suboptimal function composition implementation can lead to high cost and slow performance [5].

4) Application Development Challenges: As previously discussed, serverless FaaS platforms mainly focus on mitigating runtime complexities related to operating and managing FaaS functions. They typically offer insufficient programming support, which is often limited to rudimentary programming models, such as triggering a FaaS function with an universal event. Unfortunately, this leaves a significant gap, putting a lot of burden on developers:

- Misconfigurations of support services and infrastructure resources due to suboptimal defaults.
- Insufficient error handling mechanisms.
- Limited type safety support for function composition.
- Testing the functions beyond the unit tests, e.g., integration testing.
- Concurrency management and transactions [19]

5) AI & EI Challenges: We distinguish between *intelligence for the Edge* and *intelligence at the Edge*. The former concerns how we can apply AI methods to solve problems for which handcrafted solutions would be infeasible or of poor quality, such as constrained optimization problems. The latter, that is intelligence at the Edge addresses how we can deploy AI & EI models for intelligent tasks running in the Edge-Cloud continuum, such as voice recognition. Challenges concerning both problems are related to the heterogeneity of the necessary AI accelerators to execute models, i.e., to provide the right resources for the various tasks and environments.

- SLO-aware model selection in the ample solution space.

- Dealing with model execution overhead, such as latency and bandwidth consumption.
- Optimal splitting of the inference tasks across Edge-Cloud continuum.
- Native access to specialized hardware (e.g., AI inference accelerators) and core AI libraries.

III. Serverless Computing Fabric

A novel software-defined, intelligence-driven, and Internet-centric solution is required to address the challenges of the serverless Edge-Cloud systems. We refer to such solution as *Serverless Computing Fabric (SCF)*. Next, we present our vision for the novel Serverless Computing Fabric and outline its high-level reference architecture.

The SCF represents a paradigm shift from traditional services and platforms computing to a fabric-centric computing where digital resources, infrastructures, and systems become commodities, which permeate the entire computational and data continuum. The SCF advocates a seamless integration of existing and emerging computational resources.

The SCF deals with geographically dispersed and heterogeneous resources. However, rather than connecting nodes for a common goal, as many of the contemporary approaches do, in SCF the Edge-Cloud resources are democratized and exposed in a uniform manner to accommodate execution of arbitrary user-defined functions and applications across the entire Edge-Cloud continuum. Specifically, the SCF offers general-purpose resources and subsumes any cluster of nodes, regardless of its compute capacities or underlying system architecture. Additionally, SCF considers the gradient of a compute hierarchy in geographically distributed nodes.

We organize the SCF's main principles and runtime mechanisms into four key areas (Figure 3): 1) Reliability and performance engineering 2) Serverless infrastructure orchestration 3) FaaS Programming support 4) Serverless AI and EI These areas represent general focal points of SCF. More concretely, SCF aims to provide comprehensive support to foster *reliability and performance engineering* by delivering suitable runtime mechanisms, which enable elastic computing and optimal scheduling of functions in the Edge-Cloud continuum, while explicitly considering user-defined SLOs. Further, SCF provides support for *serverless infrastructure orchestration* by introducing novel design principles for intelligent and autonomous infrastructure management, as well as developing next-generation function isolation techniques, which are specifically tailored to serverless paradigm. The SCF also offers *FaaS programming support* for developing and executing serverless functions in the Edge-Cloud continuum. Most notably, we define concrete FaaS programming models and data/request routing runtime mechanisms. Finally, SCF puts specific focus on *serverless AI and EI* applications. Here, we aim to support execution of large deep neural networks at the edge, but also facilitate model selection and suitable data flow topologies.

Before we discuss these runtime mechanisms and principles in more detail in Section IV, we give an overview of SCF's high-level architecture.

A. Overview & Architecture

Figure 2, gives an overview of the high-level reference architecture for Serverless Computing Fabric. For the sake of simplicity, the architecture illustrates only the most important components of the SCF and does not attempt to capture all the components that are necessary to implement the SCF. The SCF comprise three main layers: i) *FaaS Runtime*, ii) *FaaS Platform Layer* and iii) *Infrastructure Management and Orchestration Layer*. In the continuation we discuss the main role of each of these layers.

The *FaaS Runtime* provides the support necessary to execute and manage *the SCF Functions*. The *Functions Controller* is responsible for managing the entire lifecycle of each SCF function. This includes starting the functions, scaling them to zero when there are no active requests, but also managing persistent function invocation queues for the dormant functions. Further, the Functions Controller is responsible to elastically scale the SCF Functions as we discuss in Section IV. The *Request and data routers* are responsible to deliver function invocation requests to the SCF Functions, but also to mediate the communication among the functions. Among other things, it is the responsibility of the request and data routers to determine whether a message should be delivered to a local or a remote SCF Function and to deliver such message by utilizing the lower-level layers. Finally, the FaaS Runtime needs to facilitate communication between the SCF Functions and the external services and capabilities. To this end, it defines two special interfaces: *Support services access layer* and *Host capabilities access layer*. The Host capabilities access layer exposes the necessary low-level capabilities to the SCF Functions. For example, such capabilities include file system or networking access. This layer is necessary since the SCF Functions need to be executed in strict isolation from each other, but also from the underlying host, for obvious security and performance reasons. Moreover, the Host capabilities access layer represents a single point of integration for the specialized hardware such as AI inference accelerators and low-level libraries such as TensorflowLite, which need to execute natively. The Support service access layer provides access to a variety of high-level services provided by the FaaS Platform Layer.

The *FaaS Platform Layer* provides core services, which underpin the FaaS Runtime. These services are generally grouped into two main components: the *Support Serverless Functions & Services* and *Core FaaS Platform Runtime Mechanisms* . We mentioned that the Support service access layer provides access to a variety of high-level services. These services are provided by the Support Serverless Functions & Services component. Examples of such services include serverless and edge-native Object store, Caching services, Key/Value stores and so forth. It is worth mentioning that all such services create one logical component within the SCF,

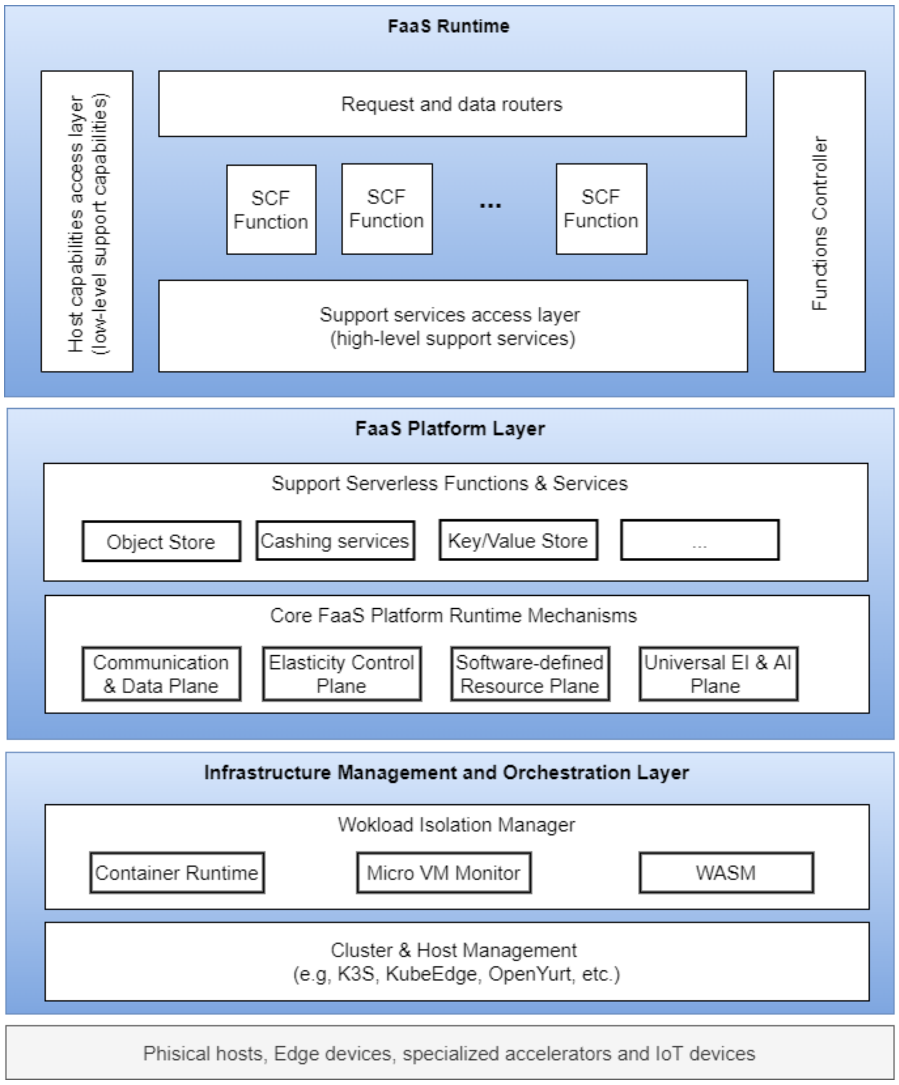

FaaS Runtime
Host capabilities access layer (low-level support capabilities)
Request and data routers
SCF Function
SCF Function
...
SCF Function
Support services access layer (high-level support services)
Functions Controller
FaaS Platform Layer
Support Serverless Functions & Services
Object Store
Cashing services
Key/Value Store
...
Core FaaS Platform Runtime Mechanisms
Communication & Data Plane
Elasticity Control Plane
Software-defined Resource Plane
Universal EI & AI Plane
Infrastructure Management and Orchestration Layer
Wokload Isolation Manager
Container Runtime
Micro VM Monitor
WASM
Cluster & Host Management (e.g, K3S, KubeEdge, OpenYurt, etc.)
Phisical hosts, Edge devices, specialized accelerators and IoT devices

Fig. 2: Serverless Computing Fabric Reference Architecture

but in reality they are typically stand-alone services, which are part of the larger serverless Edge-Cloud ecosystem. The SCF requires that all such services are built as edge-native and based on serverless (FaaS) paradigm. This is one of the key precondition to enable true serverless application and systems (see Section II). The *Core FaaS Platform Runtime Mechanisms* implement main concepts and techniques of the SCF. In Figure 2, we only show the most important components, which comprise the SCF's core FaaS platform and include: *Communication & Data Plane*, *Elasticity Control Plane*, *Software-defined Resource Plane* and *Universal EI & AI Plane*. These mechanisms are described in more detail in Section IV.

The *Infrastructure Management and Orchestration Layer* is responsible for managing the Edge-Cloud infrastructure resources. Specifically, the *Workload Isolation Manager* is responsible for isolating and orchestrating workloads which wrap the SCF Functions. The Workload Isolation Manager guarantees that different instances of the function runtime are sandboxed and isolated from each other, but also that the functions are isolated from the underlying hosts. To this end, the Workload Isolation Manager offers a novel Edge-native function isolation approach (see Section IV). It utilizes well-established solutions such as container runtimes and WebAssembly (WASM) for partitioning host's resources and limiting their usage. Finally, it offers support for fine-grained usage billing, based on used CPU cycles for each function execution. The *Cluster & Host Management* component is in charge of the deep infrastructure, i.e., it manages host nodes, which comprise the resource clusters. In the reference architecture of the SCF, we do not make any particular assumptions regarding this component. Instead, the SCF relies on the existing cluster management support and off-the-shelf solutions such as K3S or KubeEdge.

IV. Design Principles & Main Enablers of Serverless Computing Fabric

In the following, we introduce key design principles and core enabling runtime mechanisms of SCF (Figure 3). The presented mechanisms and principles are not meant to represent and exhaustive list, but rather aim to serve as an outline of our research road map towards the uniform Serverless Computing Fabric for Edge-Cloud continuum.

A. *Elastic Computing for FaaS at Edge*

Cloud-native serverless platforms rely on commodity infrastructure, small footprint, and short execution duration, combined with statistical multiplexing of a large number of heterogeneous functions over time. Elasticity at the Edge implies many challenges which are not present in the Cloud. These are caused mostly by different nature of the underlying infrastructure (scale, geographical dispersion, etc.), the topology of network connectivity and locality-awareness.

To enable the elasticity of the serverless computing fabric, we envision several disruptive changes in the contemporary perspective on elasticity. This includes both the edge-native approaches where the focus mainly lies on task-specific offloading to the cloud, as well as cloud-native approaches, which to date remain fairly limited. The latter are primarily based on a notion of autoscaling groups/sets of resources where a user specifies desired cardinality. Main requirements of elastic SCF include:

- Support for synchronizing elastic scaling actions across multiple decentralized serverless functions, e.g., serverless workflows and defining consistent and congruent scaling strategies.
- Facilitating multidimensional elasticity paradigm, which makes it possible to go beyond basic resource elasticity and address other elasticity dimensions, which include cost and quality, but also energy efficiency and carbon emission.
- Enabling rapid and uniform scaling of serverless functions across the entire Edge-Cloud continuum, by pushing the support for autonomous decision making down to a node level and facilitating distributed consensus.

To shift from current task-specific computational offloading to decentralized, consensus-based and multidimensional elasticity, SCF needs to provide novel coordination, control, and orchestration approaches that enable Edge-Cloud systems to adapt dynamically to varying load patterns and disruptive behavior in a dependable manner. Finally, a level of intelligence needs to be embedded into the SCF, which would mitigate the complexities of the mundane and error-prone tasks, which are typically required in current orchestration and coordination approaches for elasticity.

B. *Intelligent & Autonomous Infrastructure Management*

The Serverless Computing Fabric aims to build on top of the existing infrastructure virtualization management approaches (container runtimes, VMMs etc.) and cluster management (K3S, KubeEdge, OpenYurt, etc.) solutions. We intend to develop novel mechanisms for intelligent and autonomous management of the Edge-Cloud infrastructure at scale. Moreover, SCF also ties into the existing orchestration solutions, extending them with the necessary mechanisms. Due to dynamicity, heterogeneity, geographical distribution and the sheer scale of the Edge infrastructure, traditional human- and policy-driven infrastructure management and provisioning approaches (e.g., SSH-ing into Edge nodes) are hardly feasible for successful operation of the serverless fabric in Edge-Cloud continuum. Main requirements for a successful operation of SCF include:

- Automated provisioning lifecycle, together with no-code provisioning support based on off-the-shelf infrastructure components. This is a key precondition for meeting the zero operations, which is one of the main promises of serverless computing.
- Providing a high degree of autonomy to both Edge and Cloud nodes, be it physical devices or VMs, to be able to scale the provisioning processes, but also to reach optimal local decisions, e.g., with auction-based scheduling [6].
- Facilitating complete operations lifecycle with suitable AI models, which can guarantee making rapid and universally optimal resource provisioning decisions.

To make the necessary shift from traditional policy-driven operations to a fully automated, autonomous and intelligence driven operation of SCF, we need to rethink the management of the high-level infrastructure components such as containers, which underpin execution of serverless functions, but also provide mechanisms for managing deep infrastructure nodes (physical or virtual). Such mechanisms must be specifically tailored for the Edge-Cloud continuum. The SCF aims to develop suitable resource abstraction mechanisms, which represent both infrastructure nodes and higher-level infrastructure components in a uniform manner and expose them as software-defined resource units via well-defined APIs. Moreover, we advocate development of true AIOps based orchestration and coordination solutions for intelligent infrastructure provisioning across the entire Edge-Cloud resource pool. This will ultimately result in refactoring the Edge-Cloud infrastructure into autonomous and highly customizable resource components capable to cater to the diverse needs of serverless functions.

C. *Edge-native Function Isolation Techniques*

The ability to run FaaS functions in computational isolation is the key assumption of serverless computing. Computational isolation has multiple facets which typically include:

- Resource isolation, meaning any resource available to a function, such as memory or CPU can be controlled and it can be limited what such functions can access during their runtime.
- Performance isolation between functions, meaning that each function can assume having consistent and predictable performance delivered by the underlying infrastructure, regardless of its placement.

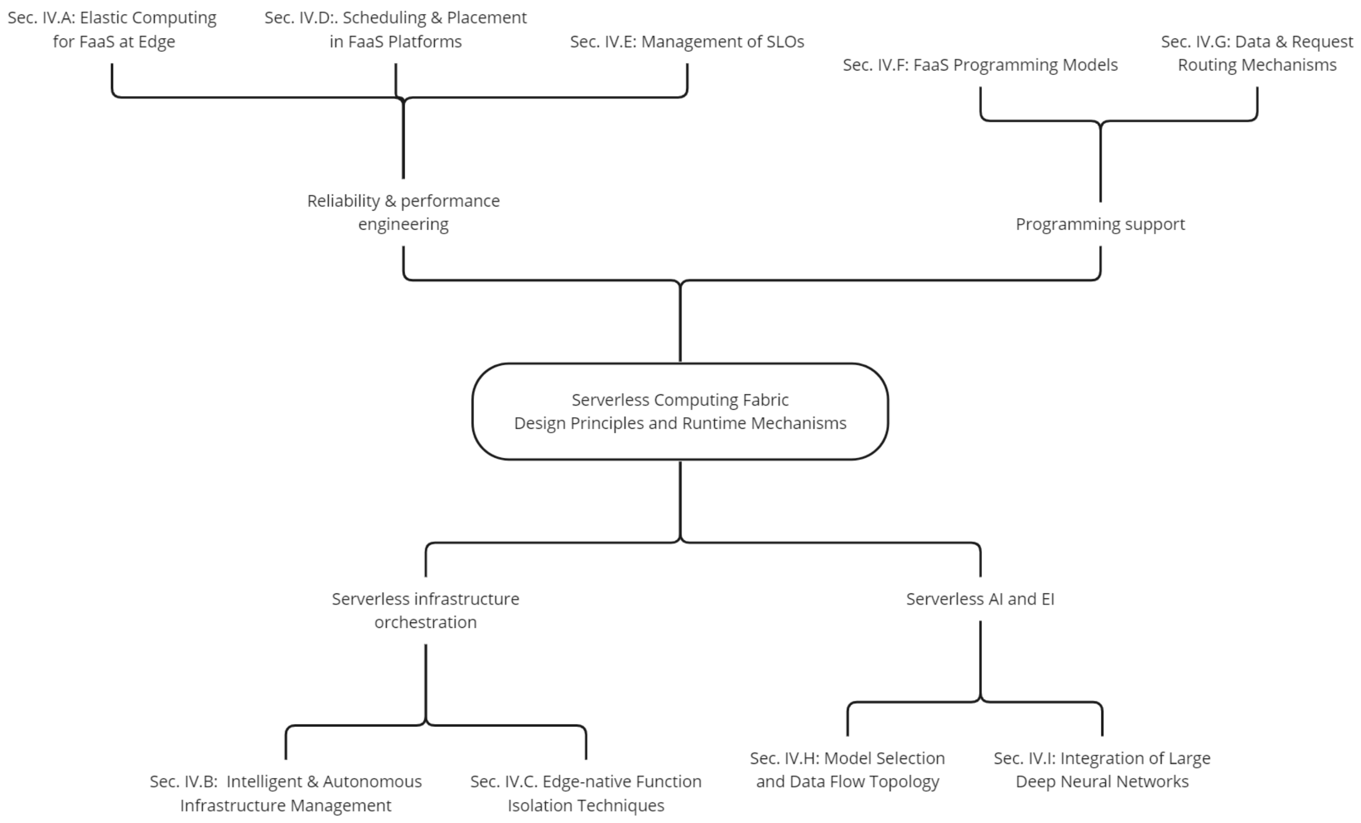

Sec. IV.A: Elastic Computing for FaaS at Edge
Sec. IV.D:. Scheduling & Placement in FaaS Platforms
Sec. IV.E: Management of SLOs
Sec. IV.F: FaaS Programming Models
Sec. IV.G: Data & Request Routing Mechanisms
Reliability & performance engineering
Programming support
Serverless Computing Fabric Design Principles and Runtime Mechanisms
Serverless infrastructure orchestration
Serverless AI and EI
Sec. IV.B: Intelligent & Autonomous Infrastructure Management
Sec. IV.C. Edge-native Function Isolation Techniques
Sec. IV.H: Model Selection and Data Flow Topology
Sec. IV.I: Integration of Large Deep Neural Networks

Fig. 3: Overview of Serverless Computing Fabric Design Principles and Runtime Mechanisms

- Software fault isolation [43], [28] sometimes referred to as “sandboxing”, means that a function is isolated form other functions in terms of exploitable bugs, malicious code, and any other software faults, which can lead to a compromised security of that function.

At the same time, computational isolation can take many forms. Well known forms include virtualization (VMs and recently Micro VMs) [1], containerization (processes, containers, pods and recently distroless and from scratch containers), microkernels and isolates [3]. Most of these approaches successfully deliver only the resource isolation. While the VM-based virtualization does address multiple facets of the computational isolation, due to its significant resource overhead it is practically prohibitively expensive for isolating functions at resource-constrained Edge.

To provide adequate support for the multi-faceted computational isolation, SCF aims to develop a novel function isolation model, which is specifically-tailored to account for the properties of Edge-Cloud continuum. Among other things the SCF FaaS isolation model has to guarantee strong resource isolation between functions, going beyond the traditional resources such as memory and CPU to also account for the attached resources such as AI accelerators, sensors and actuators. Further, it has to provide a sandboxed environment for the functions, so that they cannot interfere with the underlying host and its resources such as file system and network access. Finally, the SCF FaaS isolation model has to facilitate native execution of specific AI libraries, which require native-execution due to their performance requirements.

One of the promising approaches for edge-native function isolation is WebAssembly. It offers several features, which are particularly well suited for addressing serverless function isolation in Edge-Cloud continuum. For example, linear memory combined with control flow integrity guarantees much safer code execution compared to traditional container-based isolation. Despite receiving a lot of attention recently, WebAssembly alone cannot satisfy all the isolation requirements as described above. Therefore, SCF intends to build on top of WebAssembly and extend it in several directions, which include: 1) The SCF aims to provide a FaaS runtime that includes capabilities, which are specifically-designed to enable serverless execution of AI workloads in Edge-Cloud continuum. For example, supporting access to GPUs or specialized AI inference accelerators. 2) Integrating the SCF FaaS runtime with deep infrastructure mechanisms. 3) Making the serverless functions first-class citizens at the runtime level, in such a manner that the FaaS runtime has an explicit awareness of the running serverless functions and their (non-)functional requirements. 4) Making the SCF FaaS runtime context- and infrastructure-aware, so that is can adapt to the dynamic nature and frequent changes in the underlying infrastructure, providing more consistent experience to the serverless functions.

D. Scheduling & Placement in FaaS Platforms

The short lifetime of serverless functions compared to traditional tasks leads to a significant increase in instances that need to scheduled. Such a large number of FaaS tasks paired with the massive number of nodes in the Edge-Cloud continuum results in a scheduling load that commonly used monolithic or shared state schedulers are not designed to handle [38]. Even multi-level schedulers, such as Mesos [17] and YARN [42], often have a centralized component that might become a bottleneck in the Edge-Cloud continuum. Additionally, among other things, a high number of scheduling requests implies that each such request must be processed with a low end-to-end latency to ensure that the scheduling queue does not overflow. Other common issues in the placement of FaaS tasks, which can heavily influence the performance of functions and which are exacerbated by an Edge environment, are accounting for the distance between a function's compute node and its data, as well as, dependencies among functions that are executed in succession.

To enable efficient scheduling of FaaS tasks in the Edge-Cloud continuum and guarantee the desired execution performance of the scheduled tasks, the requirements for the SCF scheduler include:

- Leverage a distributed scheduling approach to distribute the high scheduling workload among a variable number of scheduler instances. This entails finding suitable mechanisms for scheduler state synchronization and avoidance of scheduling decision collisions (i.e., when two schedulers try to claim the same resource).
- Treat proximity to the data as a first-class citizen scheduling requirement, similar to CPU and memory resource requirements. This will allow fast data flows to and from the FaaS functions.
- Consider dependencies among functions that are part of a single workflow and schedule their instances in proximity to each other.
- Leverage network QoS requirements defined for a function's data and/or workflow dependencies to further optimize the placement. Such network QoS requirements may be explicitly defined or derived from data proximity requirements and workflow dependencies.
- Support context-aware constraints to take advantage of the heterogeneity of the Edge environment, such as geo-location, battery level, and device movement.

While the proximity to data and to other function instances from the same workflow may be treated as hard or soft constraints during scheduling, they present important aspects, which also call for appropriate abstractions, such as service graphs that extend currently available function composition graphs, to allow users to model these dependencies, such that they can be leveraged by the scheduler. To realize the scheduler for SCF we will build upon our existing SLO-aware scheduler [31] and redesign it to leverage a distributed, highly scalable architecture, whose aim will be to combine a node sampling approach, like the one found, e.g., in Sparrow [27], with an auctioning-based approach, inspired by approaches like AuctionWhisk [6]. Furthermore, we aim to introduce additional context-aware and FaaS-specific constraints, like the ones discussed above.

E. Management of SLOs

While SLOs are heavily used to guide the elasticity of traditional Cloud applications, it is challenging to enforce SLOs on FaaS tasks, due to their short lifetime. Serverless databases, which have a longer lifespan, may benefit from SLOs commonly found in Cloud computing, such as those realized by the Polaris project [30], [29]. However, for short-lived serverless functions it is hard or even impossible to adjust the resources or configuration parameters during the runtime of an existing function instance.

Thus, SLO management for FaaS needs to take an alternative approach that relies on insights gained from previous function executions and apply them when creating new function instances. Based on this observation we derive the following major requirements for SLOs in the SCF:

- Monitor the execution of function instances to determine their SLO-compliance, categorized by the function's input parameters. Based on these observations, the SCF can infer configurations that will yield the desired performance for a given set of input parameters.
- Enforce the SLOs by applying the previously gained knowledge on the instantiation of new functions.
- Leverage network QoS SLOs defined for a function (either explicitly or implicitly through workflow dependencies and input data requirements) to optimize its placement in the Edge-Cloud continuum to ensure that performance SLOs are met.

While the enforcement of network QoS SLOs needs to be tackled by the scheduler [31], the enforcement of almost all other SLOs will occur at the creation of new function instances. This entails a deep integration of SLOs into the orchestration layer of SCF and will benefit heavily from the use of AI models to predict the performance of a function, based on its input parameters. The monitoring data gained from previous executions can be used for profiling the function and create an accurate model for its performance. When a new instance of a function is created, the SCF orchestration layer will look up the most performant configuration, based on the current input parameters and use this configuration to instantiate the function.

F. FaaS Programming Models

The prevalent programming model for serverless functions is event-driven. Functions receive an input and return an output based on triggers, while the underlying infrastructure is abstracted away from the developers [23]. Current FaaS offerings support a variety of triggers, including HTTP requests, queues, file & database changes [37]. This limited expressiveness significantly restricts function developers [4].

The SCF is characterized not only by a heterogeneous infrastructure, but also by heterogeneous applications that require more sophisticated programming models than the simple event-driven model that is currently employed. The SCF decides to trade some transparency regarding function management and execution, in favor of optimally managing deployments to satisfy user requirements and SLOs.

We propose balancing the responsibilities and concerns between developers and the underlying runtime. Specifically, an extended programming model can expose specific tools to the developers to enable them to add additional context to the functions. The SCF can use this context to efficiently manage deployments optimally while exposing infrastructure concerns in a minimally invasive way to developers. In our vision, context can act as requirements and fundamentally support runtime mechanisms to manage function deployments. Specifically the following requirements are crucial for an optimal deployment: cost efficiency, energy efficiency, latency sensitivity, privacy concerns and location requirements. Developers can specify which requirements are important for their functions and the SCF can optimize toward them. The SCF requires intelligent and capable runtime mechanisms to monitor and enforce these requirements. Monitoring the SCF is challenging due to the dynamic nature of the infrastructure and connections. Reliable middleware components (i.e., message brokers) are not only important for request routing but also to ensure that runtime mechanisms do not act on faulty or missing data. The SCF also requires intelligent runtime mechanisms to enforce these requirements. For example, an optimization approach can use monitoring data to schedule applications on nodes with the lowest energy consumption and close to the users. Additionally, pro-active scaling approaches can predict locations in which instances are needed in the near future to avoid cold startups and minimize latency. This approach violates the spirit of Serverless Computing to hide all infrastructure details. Therefore programming model extensions must be careful and balance between exposing and hiding details from developers.

Moreover, stateful applications play an essential role in Edge-Cloud continuum and need to be explicitly supported. Stateful serverless approaches [7] have shown that the actor pattern is a suitable abstraction [8]. The SCF should offer methods to make the distinction and offer essential features (i.e., checkpoints and failure recovery).

Further, exploiting ideas of the Functional Programming (FP) paradigm can benefit developers and platform providers alike [19]. The following features of FP can benefit serverless development and tackle many challenges around reliability and development. First, FP languages offer different composing functions that can be used to generate complex workflows and abstract infrastructural concerns from the developer (i.e., the actual function composition implementation). Second, FP heavily relies on types and functions and treats them as first-class citizens. In our vision, we want to push forward a strong type system that makes FP programs safe and reliable. Types describe functions and can test the compatibility of function compositions. Third, concurrency, transaction management, failure recovery, and preemptive termination pose challenges in terms of reliability and development and FP inspired approaches can significantly push toward a development experience that makes aware of these issues and offers integrated solutions [19]. FP constructs can help wrap these cross-cutting concerns around functions without taking away the freedom of developers to solve these issues. Still, they also can be done without developer intervention.

G. Data & Request Routing Mechanisms

Interaction patterns of serverless Edge-Cloud applications are manifold. For example, human clients may sporadically access serverless applications (e.g., ordering a taxi) or continuously send requests (e.g., mobile augmented reality) to the platform over ingress controllers. IoT sensors periodically send heterogeneous data over topic-based message brokers. The requirements for implementing these routing mechanisms are manifold: 1) application developers need mechanisms to specify complex workflows (i.e., function compositions), 2) the platforms need mechanisms for autonomously managing routing components, and 3) reliable service meshes are crucial for providing suitable routing mechanisms.

Current FaaS platforms (e.g., Amazon, Google, IBM OpenWhisk) offer tools to compose workflows and related programming models (i.e., dataflows [13]) are based on the composition of multiple applications and form a directed acyclic graph (DAG). DAGs enable the SCF to reason about the data flow and routing. DAGs can give insight into the location and network requirements of applications can be analysed for dependencies. Specifically, the structure of DAGs can be analysed towards benefits of co-locating certain applications or avoiding multi-tenant situations at all cost due to competing resource requirements. These requirements can aid the routing of data and requests and give the SCF's runtime mechanism the necessary context to meet expectations.

To guarantee the aforementioned requirements, the SCF has to autonomously adapt routing components, such as load balancers and message brokers. As edge-cloud applications will likely generate an unprecedented amount of data, these components not only serve to satisfy customers but also prevent the congestion of metropolitan networks [36].

Serverless functions are most often short-running tasks and are scaled out and in based on demand. Therefore, the platform has to offer mechanisms to automatically adapt routing components based on the availability of function instances. Reliable routing becomes even more critical when platforms usually scale to zero, meaning that no function instances are running, making a reliable intermediate persistence necessary to route requests successfully. While these approaches are already established in cloud-centric systems, the Edge-Cloud continuum poses new challenges that have yet to be solved [39]. Based on the requirements above, the SFC requires a sophisticated network mesh that consists of routing components (i.e., message brokers, load balancers, and API gateways), is capable of dynamically managing them (i.e., spawning more brokers in a region with a high number of clients), offers methods to understand developer requirements, and is built on reliable and robust engines that can handle the dynamic environment of Edge-Cloud systems.

H. Model Selection and Data Flow Topology

In addition to packaging and deploying the core business logic as functions, intelligent applications which rely on tasks such as voice recognition or object detection must select appropriate machine learning models. A selection process is non-trivial since models can perform the same tasks while varying in size, inference latency, capacity, and hardware optimization. Rather than forcing machine learning engineers to search for the appropriate model from an exponential search space, model selection automates this process.

Model selection has seen considerable research in cloud computing, where state-of-the-art systems abstract the entire selection process, allowing clients to specify AI tasks coupled to desired SLOs [34]. However, model selection for the edge is distinct from the cloud and substantially more challenging for two reasons. First, the economies of scale incline cloud providers to specialize in a specific subset of chip architectures. In contrast, our vision of an SCF needs to integrate the entire spectrum of available AI accelerators. Moreover, even when considering devices with the same chip architecture, profiling models for latency and energy consumption is less straightforward as it depends on a device's available resources. Second, a model's input modality impacts a scheduler's strategy to uphold SLOs where the correct data flow topology and model placement is essential for modalities that require a significant amount of bandwidth. For example, to comply with latency-related SLOs, sending a visual feed over the network requires more bandwidth than textual input, giving visual models more priority to the scarce resources close to the source.

I. Integration of Large Deep Neural Networks

Deep Neural Network (DNN) models which can reliably execute inference tasks incur high resource consumption on most available hardware. Accordingly, academia and industry dedicate much attention to conceiving solutions to accommodate local inference tasks in constrained environments. They range from devising novel compact architectures [35] to altering existing architectures via quantization, pruning, or Knowledge Distillation [11]. An inherent shared trait of all such approaches is their performance and model capacity trade-offs. Alternatively, mobile applications can offload the inference tasks to a cloud server where we assume unlimited resources capable of fast inference regardless of load. Here, the bottleneck is transferring the input data to a remote server. Especially for visual tasks, there remains one obvious caveat. Continuous high-dimensional data streams must compete for limited bandwidth, which can incur unacceptable end-to-end latency caused by network congestion. More recently, Split Computing has emerged as a paradigm and middle ground between offloading the task to a powerful remote model and onloading it to a weaker embedded device [25]. As illustrated in Figure 4, the idea is to split a model into a small local head and a large remote tail. The head performs light feature extraction before sending the intermediate results to the remote tail model. Note that the head is deployed on a constrained end device, i.e., it can only contain a limited number of layers, which must downsample the original input such that the intermediate results are significantly smaller than the initial input. Hence, finding the right balance is challenging because the combined time for executing the head model, transferring the intermediate results, and executing the tail model must be shorter than offloading and executing the input task at the unmodified monolithic model.

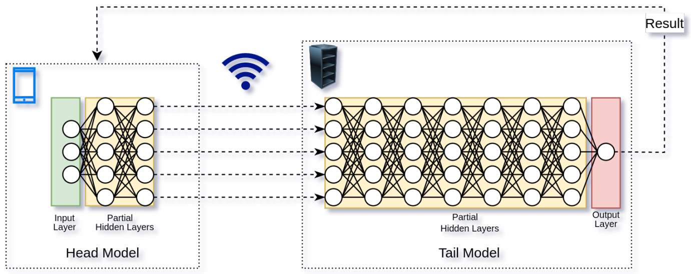


Fig. 4: Split Computing Concept

To successfully accommodate demanding AI applications in our SCF and to fully exploit the available but distributed resources, we plan on a serverless abstraction as future work [14]. However, existing approaches do not meet the requirements to enable split computing for serverless applications. Specifically, we require a runtime that can dynamically decide the offload point for each input and coarse-grained layer. Additionally, the scheduler needs to be lightweight enough such that the dynamic split decision does not outweigh the benefit of a joint on- and offload execution. To this end, we build a runtime system for dynamic split points and conceive a programming model that allows clients to organise DNNs into coarse-grained blocks so that a scheduler can deploy the functions with all or only a subset of the blocks. Then, the input is processed by a series of recursive function calls, i.e. each call executes one block taking the result of the previous block as its input.

The advantage of such an abstraction is twofold. First, we can decouple the decision-making process for onloading, offloading, and model splitting from the client with a background runtime system. Second, such an abstraction allows us to define model execution at a layer level, i.e., each layer is a possible split point enabling a scheduler to decide when to offload dynamically without the constraints of a single static split point.

V. Related Work

The related work is split into two categories. First, we compare our vision with current serverless edge platforms and highlight missing features and differences to our vision.

Afterward, we discuss works that give an overview of future Serverless Edge platforms and general issues.

Kjorveziroski et al. [22] present a survey of Serverless Edge platforms with a focus on IoT. The systematic comparison of existing approaches focuses on implementations, efficiency, scheduling, benchmarks, security aspects, and source code availability. Most of the approaches focus only on one or few aspects but no surveyed work fulfills all categories. Further, the survey has shown that Serverless Edge platforms are not limited to a specific compute environment (i.e., Edge Computing) but try to offer versatile products suiting the Edge-Cloud continuum. Nevertheless, first commercial attempts are limited to a specific environment. Ioini et al. [18] also perform a structured survey on Serverless Edge platforms. They focus on comparing existing commercial and open source platforms while looking into categories such as high availability, portability, programming language support, investment cost, and AI support. The survey leads them to identify issues surrounding vendor lock-in, lack of a decision framework when to use serverless and lack of best practices, patterns and anti patterns. Our work tackles some of those issues and gives guidelines for programming models and SLO identification while highlighting open issues for best practices (i.e., AI deployment). Fortier et al. [13] present Dyninka, a platform that combines FaaS and distributed dataflow applications. Their approach explores the possibility of combining dataflow programming and Serverless Computing to create a novel platform with an emphasis on developing applications. The system uses container-based virtualization and is integrated into Kubernetes and uses multitier programming to let developers specify interactions between individual components (i.e., functions). Other works have investigated and proposed solutions with preliminary results [9], without dedicated AI support or an extended programming model [40], [10].

Xie et al. [45] discuss challenges and present design principles on how Serverless Computing can enable Edge Computing. The authors present a systematic overview of Serverless Edge Computing networks and propose an architecture. The architecture is split into multiple layers and the authors describe in detail the components and interactions. In contrast to them, we propose mechanisms for SLOs, an extended programming model, and key challenges for Edge Intelligence. Aslanpour et al. [2] present a high-level view on Serverless Edge Computing and highlight opportunities (i.e., pay-per-use) and open issues (i.e., cold starts) surrounding Serverless Edge Computing. Rausch et al. [32] present a serverless platform with a focus on Edge Intelligence. They discuss details of a possible programming model for Edge Intelligence and outline how current cloud-centric platforms have to adapt to the Edge-Cloud continuum based on use case examples. Our work is complementary to these papers and we extend the discussion and present new mechanisms and challenges for the SCF.

VI. Conclusion

We introduced Serverless Computing Fabric (SCF), as a novel software-defined, intelligence-driven, and Internet-centric solution that can successfully address the challenges of serverless applications in the Edge-Cloud continuum. We showed how the SCF represents a paradigm shift from traditional services and platforms computing to a fabric-centric computing where digital resources, infrastructures, and systems become commodities, which permeate the entire computational and data Edge-Cloud continuum. We presented a high-level reference architecture of the SCF and discussed its main components and core services. Finally, we introduce key design principles and core enabling runtime mechanisms of SCF. These principles and mechanisms are intended to serve as an outline of our research road map towards the uniform Serverless Computing Fabric for Edge-Cloud continuum.

Acknowledgments

This work is supported by Futurewei's Cloud Lab. as part of the overall open source initiative.

References

[1] A. Agache, M. Brooker, A. Iordache, A. Liguori, R. Neugebauer, P. Piwonka, and D.-M. Popa. Firecracker: Lightweight virtualization for serverless applications. In *17th USENIX symposium on networked systems design and implementation (NSDI 20)*, pages 419–434, 2020.

[2] M. S. Aslanpour, A. N. Toosi, C. Cicconetti, B. Javadi, P. Sbarski, D. Taibi, M. Assuncao, S. S. Gill, R. Gaire, and S. Dustdar. Serverless edge computing: vision and challenges. In *2021 Australasian Computer Science Week Multiconference*, pages 1–10, 2021.

[3] V. Authors. V8, 2008.

[4] I. Baldini, P. Castro, K. Chang, P. Cheng, S. Fink, V. Ishakian, N. Mitchell, V. Muthusamy, R. Rabbah, A. Slominski, et al. Serverless computing: Current trends and open problems. In *Research advances in cloud computing*, pages 1–20. Springer, 2017.

[5] I. Baldini, P. Cheng, S. J. Fink, N. Mitchell, V. Muthusamy, R. Rabbah, P. Suter, and O. Tardieu. The serverless trilemma: Function composition for serverless computing. In *Proceedings of the 2017 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, pages 89–103, 2017.

[6] D. Bermbach, J. Bader, J. Hasenburg, T. Pfandzelter, and L. Thamsen. Auctionwhisk: Using an auction-inspired approach for function placement in serverless fog platforms. *Software: Practice and Experience*, 52(5), 2022.

[7] S. Burckhardt, C. Gillum, D. Justo, K. Kallas, C. McMahon, and C. S. Meiklejohn. Durable functions: semantics for stateful serverless. *Proc. ACM Program. Lang.*, 5(OOPSLA):1–27, 2021.

[8] D. Charousset, R. Hiesgen, and T. C. Schmidt. Revisiting actor programming in c++. *Computer Languages, Systems & Structures*, 45:105–131, 2016.

[9] M. Ciavotta, D. Motterlini, M. Savi, and A. Tundo. Dfaas: Decentralized function-as-a-service for federated edge computing. In *2021 IEEE 10th International Conference on Cloud Networking (CloudNet)*, pages 1–4. IEEE, 2021.

[10] C. Cicconetti, M. Conti, and A. Passarella. A decentralized framework for serverless edge computing in the internet of things. *IEEE Transactions on Network and Service Management*, 18(2):2166–2180, 2020.

[11] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proceedings of the IEEE*, 108(4):485–532, 2020.

[12] 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 of Things Journal*, 7(8):7457–7469, 2020.

[13] P. Fortier, F. Le Mouël, and J. Ponge. Dyninka: a faas framework for distributed dataflow applications. In *Proceedings of the 8th ACM SIGPLAN International Workshop on Reactive and Event-Based Languages and Systems*, pages 2–13, 2021.

[14] A. Furutanpey and S. Dustdar. Adaptive and collaborative inference: Towards a no-compromise framework for distributed intelligent systems. *Proceedings of the 18th International Conference on Web Information Systems and Technologies*, 2022.

[15] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere. Edge-centric computing: Vision and challenges, 2015.

[16] A. Glikson, S. Nastic, and S. Dustdar. Deviceless edge computing: extending serverless computing to the edge of the network. In *Proceedings of the 10th ACM International Systems and Storage Conference*, pages 1–1, 2017.

[17] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11)*, Boston, MA, 2011. USENIX Association.

[18] N. E. Ioini, D. Hästbacka, C. Pahl, and D. Taibi. Platforms for serverless at the edge: a review. In *European Conference on Service-Oriented and Cloud Computing*, pages 29–40. Springer, 2020.

[19] A. Jangda, D. Pinckney, Y. Brun, and A. Guha. Formal foundations of serverless computing. *Proceedings of the ACM on Programming Languages*, 3(OOPSLA):1–26, 2019.

[20] J. Jiang, S. Gan, Y. Liu, F. Wang, G. Alonso, A. Klimovic, A. Singla, W. Wu, and C. Zhang. Towards demystifying serverless machine learning training. In *Proceedings of the 2021 International Conference on Management of Data*, pages 857–871, 2021.

[21] E. Jonas, J. Schleier-Smith, V. Sreekanti, C.-C. Tsai, A. Khandelwal, Q. Pu, V. Shankar, J. Carreira, K. Krauth, N. Yadwadkar, et al. Cloud programming simplified: A berkeley view on serverless computing. *arXiv preprint arXiv:1902.03383*, 2019.

[22] V. Kjorveziroski, S. Filiposka, and V. Trajkovik. Iot serverless computing at the edge: A systematic mapping review. *Computers*, 10(10):130, 2021.

[23] H. Lee, K. Satyam, and G. Fox. Evaluation of production serverless computing environments. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pages 442–450. IEEE, 2018.

[24] Y. Matsubara, M. Levorato, and F. Restuccia. Split computing and early exiting for deep learning applications: Survey and research challenges. *ACM Computing Surveys (CSUR)*, 2021.

[25] Y. Matsubara, M. Levorato, and F. Restuccia. Split computing and early exiting for deep learning applications: Survey and research challenges. *ACM Computing Surveys (CSUR)*, 2021.

[26] S. Nastic and S. Dustdar. Towards deviceless edge computing: Challenges, design aspects, and models for serverless paradigm at the edge. In *The Essence of Software Engineering*, pages 121–136. Springer, Cham, 2018.

[27] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica. Sparrow: Distributed, low latency scheduling. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, New York, NY, USA, 2013. Association for Computing Machinery.

[28] G. Peach, R. Pan, Z. Wu, G. Parmer, C. Haster, and L. Cherkasova. ewasm: Practical software fault isolation for reliable embedded devices. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(11):3492–3505, 2020.

[29] T. Pusztai, A. Morichetta, V. C. Pujol, S. Dustdar, S. Nastic, X. Ding, D. Vij, and Y. Xiong. A novel middleware for efficiently implementing complex cloud-native slos. In *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*, 2021.

[30] T. Pusztai, A. Morichetta, V. C. Pujol, S. Dustdar, S. Nastic, X. Ding, D. Vij, and Y. Xiong. Slo script: A novel language for implementing complex cloud-native elasticity-driven slos. In *2021 IEEE International Conference on Web Services (ICWS)*, 2021.

[31] T. Pusztai, S. Nastic, A. Morichetta, V. Casamayor Pujol, P. Raith, S. Dustdar, D. Vij, Y. Xiong, and Z. Zhang. Polaris scheduler: Slo- and topology-aware microservices scheduling at the edge. In *2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC)*. 2022.

[32] T. Rausch, W. Hummer, V. Muthusamy, A. Rashed, and S. Dustdar. Towards a serverless platform for edge {AI}. In *2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 19)*, 2019.

[33] T. Rausch, A. Rashed, and S. Dustdar. Optimized container scheduling for data-intensive serverless edge computing. *Future Generation Computer Systems*, 114:259–271, 2021.

[34] F. Romero, Q. Li, N. J. Yadwadkar, and C. Kozyrakis. {INFaaS}: Automated model-less inference serving. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pages 397–411, 2021.

[35] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[36] M. Satyanarayanan. The emergence of edge computing. *Computer*, 50(1):30–39, 2017.

[37] J. Scheuner, M. Bertilsson, O. Gronqvist, H. Tao, H. Lagergren, J.-P. Steghöfer, and P. Leitner. Triggerbench: A performance benchmark for serverless function triggers (short paper). In *2022 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2022.

[38] M. Schwarzkopf. The evolution of cluster scheduler architectures, 2016.

[39] M. R. S. Sedghpour and P. Townend. Service mesh and ebpf-powered microservices: A survey and future directions. In *2022 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, pages 176–184. IEEE, 2022.

[40] 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 *2022 IEEE 6th International Conference on Fog and Edge Computing (ICFEC)*, pages 17–25. IEEE, 2022.

[41] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella. On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration. *IEEE Communications Surveys & Tutorials*, 19(3):1657–1681, 2017.

[42] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, New York, NY, USA, 2013. Association for Computing Machinery.

[43] R. Wahbe, S. Lucco, T. E. Anderson, and S. L. Graham. Efficient software-based fault isolation. In *Proceedings of the fourteenth ACM symposium on Operating systems principles*, pages 203–216, 1993.

[44] B. Wang, A. Ali-Eldin, and P. Shenoy. Lass: running latency sensitive serverless computations at the edge. In *Proceedings of the 30th International Symposium on High-Performance Parallel and Distributed Computing*, pages 239–251, 2021.

[45] R. Xie, Q. Tang, S. Qiao, H. Zhu, F. R. Yu, and T. Huang. When serverless computing meets edge computing: architecture, challenges, and open issues. *IEEE Wireless Communications*, 28(5):126–133, 2021.

[46] L. Zhao, Y. Yang, Y. Li, X. Zhou, and K. Li. Understanding, predicting and scheduling serverless workloads under partial interference. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15, 2021.