IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 18 July 2023, accepted 14 August 2023, date of publication 21 August 2023, date of current version 5 September 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3307026

PERSPECTIVE

Dynamic, Context-Aware Cross-Layer Orchestration of Containerized Applications

RUTE C. SOFIA[1], (Senior Member, IEEE), DOUG DYKEMAN[2], PETER URBANETZ[2], AKRAM GALAL[1], (Member, IEEE), AND DUSHYANT ANIRUDHDHABHAI DAVE[1]

[1]fortiss GmbH, Research Institute of the Free State of Bavaria for Software Intensive Systems and Services, 80805 Munich, Germany
[2]Zurich Research Laboratory, IBM Research Europe, 8803 Zürich, Switzerland

Corresponding author: Rute C. Sofia (sofia@fortiss.org)

This work was supported in part by the Horizon Europe Cognitive Decentralised Edge Cloud Orchestration (CODECO) under Grant 101092696, and in part by the fortiss-IBM Center for Artificial Intelligence (C4AI) EDGE.

ABSTRACT Container orchestration handles the semi-automated management of applications across Edge-Cloud, providing features such as autoscaling, high availability, and portability. Having been developed for Cloud-based applications, container orchestration faces challenges in the context of decentralized Edge-Cloud environments, requiring a higher degree of adaptability in the verge of mobility, heterogeneous networks, and constrained devices. In this context, this perspective paper aims at igniting discussion on the aspects that a dynamic orchestration approach should integrate to support an elastic orchestration of containerized applications. The motivation for the provided perspective focuses on proposing directions to better support challenges faced by next-generation IoT services, such as mobility or privacy preservation, advocating the use of context awareness and a cognitive, cross-layer approach to container orchestration to be able to provide adequate support to next-generation services. A proof of concept (available open source software) of the discussed concept has been implemented in a testbed composed of embedded devices.

INDEX TERMS Context-awareness, IoT, edge computing, machine learning, data observability.

I. INTRODUCTION

Internet-based services, and in particular *Internet of Things (IoT)* services provide a way to exploit data across different vertical domains to reach a higher degree of efficiency. Up until recently, IoT data processing and storage has been mostly based on Cloud solutions, which implied transmitting all the collected data from IoT devices to the Cloud. However, the increasing amount of generated data and the frequent exchange of such data with the Cloud brought in new challenges, such as delay in processing data, increase in energy consumption and costs [1]. Edge computing [2] is a paradigm that can assist in overcoming some IoT challenges, by pushing the computation and data processing ''closer'' to the end user or to field-level devices. This decentralization of computation and networking functions in the so-called *Edge-Cloud continuum* [3] envisions the distribution of computation, data, storage, and application logic often across multiple operational regions controlled by different service operators. Today, the spread of different functions across the Edge-Cloud IoT device is made possible through software- and hardware-based virtualization solutions that provide support to easily setup and deploy applications based on micro-service architectures [4]. The most popular software-based virtualization approaches considered in the context of the Edge-Cloud continuum are *virtual machines (VM)*, managed by and *virtual machine monitors (VMM)*, also known as *hypervisors*, and container technologies, such as Docker [5], [6], [7]. Container technologies and hypervisors provide the basis to run applications efficiently by supporting isolation of applications within the system (software and hardware). The key differences between these technologies are related to the level of isolation provided. When considering a container solution such as Docker, the application is isolated via containerized images, but still shares the same kernel with different containers on the same host. Hence, processes running inside containers that compose an application can be accessible to the host system, assuming that

The associate editor coordinating the review of this manuscript and approving it for publication was Derek Abbott.

privileges for such an application exist. VM approaches make anything running inside a VM independent from the host operating system. At start time, a VM boots a new dedicated kernel slice for a VM environment, and therefore activates required operating system processes, which a container-based approach does not need to have. Performance issues and comparisons between different approaches are available in the related literature [8]. The focus of this paper is on container-based technologies and, in particular, solutions that manage container technologies, known as *container orchestrators* [9].

Due to the flexibility introduced by container-based approaches, today IoT services that have been traditionally deployed on the Cloud, e.g., an IoT data analytics service, can now be easily deployed across Edge-Cloud. Container-based approaches for IoT enable the exploration of new computational models and facilitate the deployment of new business models derived from the expanded data processing capabilities from the Edge to the Cloud [10]. However, the generalized use of container technologies to support a fast and flexible deployment of IoT services across Edge-Cloud is only feasible if there is an adequate control plane, capable of managing the setup and run-time of containerized applications. Container orchestrators provide a way to reduce human intervention and increase efficiency in the overall setup and lifetime management of IoT services and applications on Edge-Cloud [11], [12], [13]. Specifically, container orchestration refers to the handling and scheduling of the *workload* (micro-service, component of an application plus its state) of individual containers for applications based on micro-service architectures. Some of the tasks handled by orchestrators include: configuration and scheduling as well as provisioning of containers; checking availability of containers; scaling the system to balance application workloads across the overall infrastructure; allocating resources to the different containers, and monitoring their health; securing the communication exchange between containers. The most popular examples of container orchestrators are *Kubernetes (K8s)*,[1] Docker Swarm (DS)[2] [14].

The current generation of container orchestrators addresses the scheduling and distribution of workload across Edge-Cloud in a semi-static fashion, relying on a *replication* approach. By replication, it is meant that the overall orchestration of containerized applications relies on an approach where the orchestrator is responsible for handling multiple replicas of the same containerized applications, or of its containerized micro-services, deciding to activate, or to stop specific replicas due to a preconfigured set of rules. This approach is semi-static, requires support by a human operator for each deployed and active application.

However, next-generation IoT and Internet services rely on multiple applications across the Edge-Cloud continuum. With the integration of computation on the Edge and in particular on the so-called *far Edge/deep Edge* [15], applications and their micro-services will be running across mobile, often constrained interconnected nodes. Hence, a high degree of variability in terms of computational and networking resources has to be supported, and next-generation container orchestrators have to be able to account for such behavior.

Consequently, the motivation that led to this work concerns the belief that in order for container orchestrators to achieve a higher degree of elasticity, required in the midst of challenges such as mobility, intermittent connectivity, and the deployment of applications across restricted devices, it is important to devise new features based upon context-awareness, learning, and adaptation capabilities.

Context-awareness in this paper refers to the ability of a system to consider knowledge about its environment Context awareness refers to the ability of a system to consider knowledge about its environment, to perform specific actions [16], [17], to perform specific actions. In this context, a system can be a computational system; a cyber-physical system; a set of cyber-physical systems. Usually, contextual information falls into a wide range of categories such as computing context (e.g., available processors, memory, nearby resources); user context (e.g., location, user profiles, nearby users); environmental context (e.g., lighting, temperature, etc.).

In the context of the perspective provided in this paper, context-awareness refers to functional and non-functional requirements provided by the application and system; from the network; from the user; and from the data. In other words, container orchestration mechanisms need to take into account a jointly devised cross-layer approach to provide adequate support to next generation IoT services and Internet services.

To achieve such a dynamic behavior, this perspective paper identifies and debates different aspects that need to be addressed, and proposes a high-level functional design for a context-aware orchestration, focusing on the following challenges:

- What kind of context-awareness, which parameters are relevant to integrate into an orchestrator?
- How to model context in a way that is relevant to be applied in the overall management of Edge-Cloud application lifetime management?
- What is the role of *Machine Learning (ML)* in the overall orchestration process? Which type of learning pattern is required?
- Is current replication enough, or are there use-cases where it would be beneficial to fully handover (*offload*) application workload and its state from one location to another?

In order to contribute to answering the aforementioned research questions, this paper provides the following contributions.

- Provides a novel perspective and promotes a debate about current container orchestration approaches and proposes steps to achieve a more dynamic orchestration behavior, better suited for next-generation IoT services across the Edge-Cloud continuum.

[1] https://kubernetes.io/
[2] https://dockerswarm.rocks/

- Proposes the use of context awareness, detailing specific indicators that can be considered, and the process to integrate context awareness into reference container orchestrators.
- Proposes an architecture for dynamic container orchestration based on ML and context awareness. integrating parameters related to application requirements, data requirements, system requirements, network requirements, and user behavior.
- Describes a first proof-of-concept (Movek8s) deployed in a testbed based on embedded devices, to assist the reader and developers in understanding the overall proposed concept.

The remainder of the paper is organized as follows. Section II describes the work related to ours, explaining our key contributions. Section III covers background on what is container orchestration; which architectural solutions are available, their advantages, and gaps. Section IV provides an overview of scenarios across different vertical domains, where container orchestration requires a more dynamic behavior than the one available today. Section V addresses the integration of ML on container orchestrators; approaches as of today; why further decentralization is required in the training and learning process, and which advantages such decentralization may bring. Section VI discusses data observability and its role in achieving a robust and dynamic container orchestration across Edge-Cloud. Section VII proposes a functional architecture for dynamic orchestration, explaining the role of context-awareness; ML-based orchestration; and how such a framework may interact with an orchestrator such as Kubernetes. Section VIII describes a proof of concept, Movek8s,[3] implemented on a test bed composed of embedded devices. This proof-of-concept helped us in understanding the implications of integrating a simple form of context-awareness (location) into a container orchestrator (Kubernetes) and is described to assist the reader in understanding how the proposed framework may be instantiated and the purpose of doing so. Section IX concludes the article and proposes a few directions for future work.

II. RELATED WORK

Current orchestrators such as K8s and DS offer fundamental features such as (i) resource control, (ii) service scheduling, (iii) load-balancing, (iv) auto-scaling, and (v) high service availability. However, they have some limitations such as lack of elasticity to handle a more variable service behaviour, or support for a higher degree of automation, i.e., *zero configuration* [25]. his section summarizes analysed related work, highlighting contributions expected by our proposal concept. To assist the reader, a summary of the analysed approaches including advantages and disadvantages is provided in Table 1.

A first category of related literature considers a more adaptive behavior, based on improvements to orchestration scheduling. For example, Bulej et al. propose a self-adapting *Kubernetes (K8s)* scheduler aimed at a better support of time-sensitive applications [18]. Their approach relies on continuous probing to assess current performance and to allow K8s to detect and react to failures, in order to satisfy bounded latency requirements. This brings in some degree of adaptation, but has the disadvantage of requiring constant probing by the system. Rossi et al. have proposed a scheduling approach that takes into account geolocation [19]. Their approach considers the application of *Reinforced Learning (RL)* to dynamically control the number of application replicas across different locations and address the distribution based on an optimization problem that takes into consideration network-aware heuristics, e.g. path or hop delay. Zhang et al. outline a predictive container autoscaling algorithm (ASARSA) [20]. ASARSA combines *Automated Integrated Moving Average (ARIMA)* and Artificial Neural Network models to schedule containers in a timely manner and improve the accuracy of scheduling decisions. Although ARIMA scales well with large-scale datasets, it is limited to linear models [26]. Similarly, Toka et al. have also described the application of ML in the scheduling process for autoscaling aspects, where the authors have provided a taxonomy for the application of ML in container orchestration [26]. In this category of related work, there is a specific focus on improving, via self-adaptation, an existing feature of K8s, often related with the support of time-sensitive applications, for instance, auto-scaling, or load-balancing. This adaptation is often based on heuristics that take into consideration functional application or networking requirements, such as latency.

Another category of related work concerns the application of context-awareness to orchestrators, so that the overall orchestration process can be adapted to existing conditions, and the resulting application deployment becomes smoother. Ogbuachi et al. have presented a debate on the use of context awareness in the context of edge-based applications on a 5G infrastructure [21]. The authors defined context as cluster and network data that should be integrated into the K8s scheduler, proposing a measure of *usability* of a node. Specifically, their algorithm, which has been shown to improve behavior in comparison to K8s, relies on collected node (e.g., CPU, memory) and network data (e.g., network delay) and current workload, as well as requirements of the applications, to propose an adaptation while preserving fairness in terms of workload distribution. Kaur et al. propose a multiobjective scheduling optimization based on integer linear programming, KEIDS, which aims to minimize energy usage for *Industrial IoT (IIoT)* environments [22]. Context is defined in their work as node energy usage, Carbon footprint emissions, and also performance interference, and their algorithm shows relevant performance improvements in terms of energy consumption efficiency in comparison to existing K8s schedulers, based on real-time Google traces. Energy and latency, along with specific node usage indicators such as CPU and memory, are the most common context indicators considered in elastic orchestration. However, there are other relevant context indicators

[3] https://git.fortiss.org/iiot_external/movek8s

TABLE 1. Advantages and disadvantages of the related work analyzed and of our proposed concept.

Related work	Pros	Cons
Self-adapting K8s scheduler for time-sensitive applications [18]	Supports bounded latency	Requires continuous active probing and just supports latency aspects, ignoring other relevant requirements from the application, data flow, or network.
K8s scheduler with knowledge of geolocation and network awareness in the form of path and hop delay [19]	Controls dynamically the number of replicas across different locations taking into account hop count and path delay	Provides a very basic form of network awareness and does not integrate other requirements.
ASARSA, predictive auto-scaler algorithm [20]	Improves the scheduling accuracy via NN models	Scales well if datasets are dense, but is limited to linear models.
Network-aware approach [21]	Provides a more flexible scheduling taking into consideration a measure of node usability, which considers both node usage and network usage aspects	Considers autoscaling aspects only, and does not support intermittent connectivity aspects.
KEIDS [22]	Supports energy awareness	Other relevant context indicators that can be sensed throughout the supported system, going beyond the network and application requirements [23] are not considered.
Advocates the need to allow K8s to support a higher degree of decentralization in terms of application workload setup and runtime to support heterogeneous Edge-Cloud environments [24]	Debates on the need to have support for decentralization in terms of application setup and runtime	Focuses on network and application requirements for a single vertical domain, Smart Cities.
Our concept	Provides support to Edge-Cloud considering application-awareness, system-awareness, network-awareness, and data-awareness, thus creating a more elastic support to heterogeneous and decentralized Edge-Cloud environments	Requires an integrated approach to a data-compute-network orchestration, which will bring a complexity trade-off that needs to be analyzed for single and federated clusters environments.

that can be sensed throughout the supported system, beyond the network and application requirements [23].

In this work, we aim at explaining why it is relevant to consider other types of parameter to define context, in particular external parameters to a system, and how these parameters can be modeled and applied in terms of container orchestration. It is our opinion that orchestration based on multiple parameters (metadata) that is collected across different OSI layers can achieve more elasticity and robustness and is expected to increase the level of efficiency and fairness of a system in terms of application workload scheduling decisions.

The proposed approach for a dynamic container orchestration to be debated in the next sections aims at supporting an heterogeneous multi-provider Edge-Cloud continuum by considering requirements derived from the application and application based parameters; system awareness (e.g., information about the computational status of nodes); network-awareness (e.g., information about specific available paths or links); data awareness (e.g., information about the status of data, such as data freshness). These aspects shall be further debated in Section VII, where our perspective on the building blocks of a dynamic container orchestration framework is debated.

The need to allow K8s to support a higher degree of decentralization in terms of application workload setup and run-time, in particular considering more variable environments across Edge-Cloud, has been the subject of a literature review in the context of Smart Cities [24]. The authors debate on the need to consider custom parameters (custom context indicators) based on networking or application requirements, to allow K8s to support Edge environments. However, it is relevant to go beyond a specific domain and to address this support in a global way, not necessarily tied to a specific domain, e.g., Manufacturing, Smart Cities, as shall be explained in Section IV.

III. CONTAINER ORCHESTRATION BACKGROUND

A. TERMINOLOGY

This subsection introduces the terminology used throughout the paper. An application is defined as being based on a micro-service architecture, where each component (*micro-service*) can be run independently based on container technology, such as Docker. This is named *containerized micro-service*, or *containerized application*. A containerized micro-service is therefore composed of the binary system, *workload*, data, and *state*, i.e., a set of global variables defined in the micro-service and required at run-time. A containerized application therefore consists of one or several containerized micro-services which are interconnected via specific *interfacing policies*. The different micro-services may run on the same device, in a decentralized way (independently, e.g., a broker and an application monitor), or in a distributed way, across different devices or virtual machines. To scale the application, it is feasible to consider transparent *replication* or *offloading*, also known as *relocation*.

Replication implies that an application and its micro-services are copied across different locations, i.e. there are

multiple replicas of the application in a system, some active, and some inactive. With replication, the decision to activate or stop an application is made based on a specific configuration and a set of policies. Replication requires a way to synchronize state across an ''old'' and a ''new'' environment where the application is deployed, and implies that even inactive applications consume resources (at least storage).

Offloading implies moving an application and its microservices across different environments. Therefore, with offloading, the ''old'' environment, the application workload, and its state are deleted and not simply made inactive; a copy of workload and of state, followed by eventual adaptation of state, is done in the ''new'' environment.

The applications supported in this process can be *stateless* or *stateful*. Stateless applications do not require data storage to work. An example is a Web search. Stateful applications keep state on clusters and require that state (data, status of the application) to be kept and eventually found.

Edge, Cloud definitions, including the notions of far Edge/near Edge follow the line of thought being driven in the European initiative *Next Generation IoT (NGIoT)*,[4] and in particular the vision for smart, decentralized Edge-Cloud environments for IoT applications [15]. Moreover, as shall be explained further in the next section, there are a few definitions used throughout the paper that relate with orchestration.

Container follows the definition of K8s, where it is a package with the overall settings (workload, state, data) to allow the execution of an application in an independent way.

A *Pod* follows also the K8s definition, being a logical wrapper entity for containers to be executed in a K8s cluster. This logical wrapper ''holds'' a group of one or more containers with shared storage and network resources, and also a common namespace, providing a definition to run the containers.

A *cluster* corresponds to the logical environment in which the pods run in a way that has been orchestrated by a human operator.

Hence, a container runs logically in a pod. A pod may hold more than one container. A cluster can hold multiple Pods (not necessarily related); Pods are grouped via logical boundaries, their namespace. Therefore, a pod is the unit of replication in a cluster.

The applications across Edge-Cloud are therefore orchestrated via multiple clusters, where an Edge environment, or an Edge-Cloud environment may be inside a single cluster (e.g., if under the operation of a same service provider) or of multiple clusters (e.g., across multi-domain environments).

B. CONTAINER TECHNOLOGY

Container technology such as Docker[5] provides a virtualization solution to isolate applications together with their state, and eventually data, and thus provides the means to run applications in a way that is independent of the underlying *Operating System (OS)* and hardware. Docker popularized the container pattern and has been an important player in the development of the underlying technology, but the container ecosystem is much broader than just Docker.

[4]https://www.ngiot.eu/

[5]https://www.docker.com/

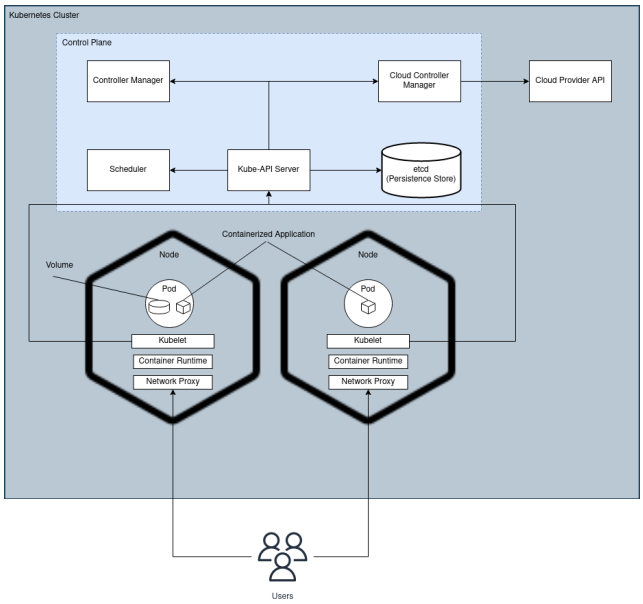

FIGURE 1. High-level perspective of the K8s architecture.

The management of containerized applications is performed via container orchestrations, as explained earlier. Container orchestrators assist in configuring and managing the overall application setup and life cycle. As has also been explained, the two most relevant container orchestrators that exist today are K8s and DS, which are explained further in the following subsections. Out of these, several other variants have been derived.

C. KUBERNETES (K8s)

K8s has been affirming itself as the de facto container orchestrator. As an open source solution, it provides support for automating the deployment of applications, as well as supporting an adequate scaling and overall run-time management. Originally developed for Cloud-based services, K8s requires some adaptation for Edge-based services.

The general management approach of K8s, for which the architecture is represented in Figure 1, relies on the concept of *Pod*, i.e., an abstraction element to manage one or more containers, with shared resources and collection of network configuration of containers that share the same IP and port space.

The pods are assigned to *worker nodes*, where a local daemon (*Kubelet*) manages the life cycle of the worker nodes. Kubelet is also the entry point into the K8s *control plane*. The control plane is therefore composed of pods that reside on the main nodes and implement *etcd*, the *scheduler*, the *controller manager* and the *API server*.

etcd is a key value internal database that represents the desired state of a cluster, based on the Raft consensus protocol [27]. It stores all the information about the pods, in which node they should run, number of instances, etc.

The *controller* is the control-plane component (control loop) that runs processes that continuously check the state of the cluster, and compare them to the desired state in etc. then being able to make or to request changes when required.

The controller usually performs rescheduling based on actions provided by the *API server*, but it can also execute the action itself. For example, a controller can scale nodes in a cluster. When the desired state is updated, the controllers will detect the mismatch and try to bring the cluster state to the desired one. K8s has some built-in controllers, but new controllers can be easily added to a cluster, for example, by running them as a pod or even outside the cluster.

The desired state of the cluster is defined by the user via the *API server*. This component receives the commands of the user and stores the desired state in etcd.

The *scheduler* (*kube-scheduler*) handles the decision on the pod (new or unscheduled) to node matching, so that Kubelet (worker nodes) can run the pods. In a first phase (*filtering*), the scheduler checks which nodes can meet the scheduling requirements. These nodes are named *feasible nodes*. In a second phase (*scoring*), the scheduler ranks the feasible nodes for the ''best'' pod deployment, calculating *scheduling priorities*, also defined in the desired state.

Since the scheduling algorithm can be very simple and since scheduling is an optimization problem, there are some ways to extend the scheduler so that it handles pod placement with finer-grained detail. The three main ways to extend the scheduler are:

- Calculating scheduling and priorities to the scheduler and recompile it from source code.
- Implement a complete new scheduler that can run instead or in parallel with the default scheduler.
- Implement a *scheduling extender*, which provides callbacks that the default scheduler calls at the end of each phase of the decision, as proposed by Santos et al. for the case of latency and bandwidth [28].

Once pods are assigned to nodes, kubelet handles the execution of pods. K8s does not actually offload application workload from one node to another; instead, it relies on replication: the K8s scheduler sends commands to kubelet on the selected nodes to start the container; and sends commands to kubelet in the old nodes, to stop containers. The containerized application state changes, however, since the container in the new location is not the same as the original. Therefore, keeping the state adequately synchronized between new and old containers requires additional configurations from the cluster administrator.

D. DOCKER SWARM

DS is the native Docker orchestration tool, currently known as Docker Swarm Mode. While K8s was first designed to manage a single cluster, DS has been devised to support the management of nodes across multiple clusters.

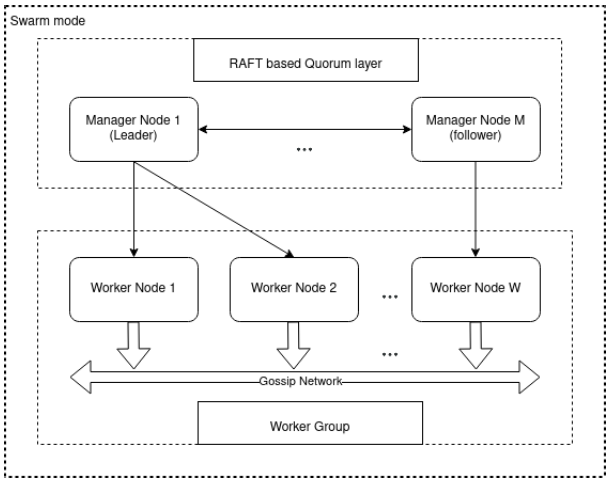

FIGURE 2. High-level perspective of the docker swarm architecture.

Similarly to K8s, the DS architecture consists of two types of nodes: *manager* and *worker* nodes. Manager nodes run in the control plane which maintains the desired state of the cluster and assign *tasks* (containers) to the working nodes, and these execute the given *tasks*. On each worker node, an agent is running and reports its internal state to the manager node, as illustrated in Figure 2.

The control plane integrates four components:

- *Orchestrator*: evaluates the state of the cluster by comparing it with the desired one and creates the tasks for each service description.
- *Allocator*: enables the assignment of tasks to worker nodes by referring to their corresponding IP address.
- *Scheduler*: receives the created tasks and checks the available nodes and its resources to decide where they will run.
- *Dispatcher*: connects to each worker node and sends task assignments to them. Each worker node periodically reports its health status to the Dispatcher.

A node manager can also be a worker node, and the robustness of the swarm can be improved by having more than one manager node. This robustness is widely known as *high availability*. The capability of having multiple manager nodes is a feature embedded in Docker Swarm, which uses a Raft implementation to maintain a distributed and consistent internal state of the entire swarm. This internal state of cluster is stored in etcd.

E. COMPARISON OF K8s AND DS

K8s and DS offer common capabilities such as a scheduler to allocate containers to nodes, high availability, state and health check of containers, and more. Both solutions use declarative languages, meaning that the human operator defines the desired state of the clusters via specific tools with some degree of flexibility and high level of interoperability. Due

to its integration into Docker, DS provides an easy way of working with its API, in contrast to K8s, which introduces its own CLI, making the learning curve steeper.

K8s implements its functionalities following a modular approach, where different functional blocks and services communicate through the API server. If new features (or third-party applications) are to be considered, then these require also integration to the API server.

Regarding performance aspects, DS performs better and adds less overhead, as shown by Nickoloff [28], who provides a performance evaluation for multiple aspects, for example, the time required to start thousands of containers in both K8s and DS considering a cluster with one thousand nodes. Pan et al. experimented with the same tools to investigate the overhead of each tool compared to containers running directly with Docker [29]. The results show a larger overhead of K8s. Beltre et al. also compare the performance of these tools and against bare metal solutions, but focused on communications in HPC applications [30]. Their analysis shows that both K8s and DS can achieve near bare-metal performance on *Remote Direct Memory Access (RDMA)* networks, when high-performance transports are enabled.

To provide a better comparison of key features in K8s and DS, and how orchestrators should evolve, Table 2 provides a descriptive comparison of features for K8s and DS, and proposes a few directions towards a novel generation of dynamic orchestrators. The proposed features are the basis for the discussion in the following sections.

F. REPLICATION AND OFFLOADING

Container orchestrators manage containerized applications based on human intervention, having as a basis replication, as has been explained earlier.

In Edge-Cloud environments, and in particular in far Edge environments, replication may create challenges, as devices are often mobile and resource constrained. Hence, it is also important to focus on the possibility of offloading and to debate and evaluate the implications of such a process in the context of dynamic container orchestration.

Replication focuses on scaling aspects: replicas of micro-services are deployed in different nodes, and can be activated or deactivated to achieve a specific performance objective.

Offloading is often used in Edge environments when there is a need, for instance, to run an application independently on the Edge to meet latency or other types of requirements. Offloading implies a load reduction in the former environment, but requires additional management to ensure stability of the overall system.

In dynamic environments, for instance, Edge-based environments involving mobile devices such as *Unmanned Aerial Vehicles (UAVs)*, *Automated Guided Vehicles (AGVs)* or even cars, there is a need not only to scale up the system, but in many cases also to scale down resource usage. Adding to this, environments that encompass multiple clusters belonging to different operators may require that security parameters be updated whenever a container is moved to a new domain, or provisioning of a dynamic and distributed trust management scheme.

With the replication model, this implies that during a scheduling process, which is usually based on a two-phase filtering and ranking scheme, the eligible nodes are selected based on available static resources at some instant in time.

Assuming a very dynamic environment, where new pods are frequently assigned to new nodes to reduce latency, then it may happen that the system may reach a point where the existing resources are not enough to meet the current system requirements. It should be highlighted that K8s never moves Pods from one node to another to free up resources. In variable mobility environments, offloading seems to be a more interesting model. However, its application may imply additional adjustments to the system, and therefore it is important to understand the offloading requirements and steps. It is also relevant to understand which indicators (beyond node usage indicators such as CPU and memory) should be passed to a container orchestrator scheduler in order to best meet the requirements of more dynamic environments. This aspect - which context indicators to consider and why - is addressed in the next subsection. It is also important to understand why offloading should occur and the impact of offloading workload and respective state at an instant in time, as well as on future system operation.

IV. GUIDING SCENARIOS

The integration of IoT services across different vertical sectors is expected to grow, backed up by the technological capability to distribute micro-services. A key enabler for this is the capability to deploy a service in a way that improves the overall system performance, which is frequently tied to latency reduction - the closer the workload is to the user or data source, the smaller the latency. This distribution of micro-services brings additional advantages, in particular for scenarios involving mobile devices. A key issue in this context is mobility of devices that are tied to the user mobility behavior. Similarly, more advanced scenarios are based on devices (IoT-Edge) that are mobile independent of users. In general, providing mobility support is an essential requirement for future orchestrators. Examples of application categories that can profit from a more dynamic orchestration are described in the remainder of this section, including:

- Time-sensitive applications, e.g., IIoT critical applications that require bounded latency and low jitter.
- Pervasive sensing applications, for example *Mobile Crowd Sensing (MCS)* applications that are time sensitive and require low energy consumption.
- Mobile content/video streaming, requiring low latency, which is supported via storage closer to the user.

A. MANUFACTURING

A worker in a factory relies on a certified device that can be used only in specific areas, to perform machine maintenance.

TABLE 2. Comparison of features between K8s and DS, and expected features in a dynamic orchestrator.

Feature	K8s	DS	Future Directions
Setup, installation and configuration	Scripting required to bring up a cluster, define the specific environment, define the internetworking between components (Pod network) and configure the dashboard.	The Docker CLI runs the different programs and, therefore, simplifies setup. The resulting clusters are not as robust as in K8s.	Better integration of a CLI and UI, to address zero-configuration.
Building and running containers	Specific API, client, and configuration (YAML) definitions.	Docker CLI allows new containers to be easily deployed.	Initial set of parameters set via a dashboard. The setting up of a cluster, adding and removing would be automated via a zero-configuration mechanism.
Logging, Monitoring	Inbuilt tools provide support to understand the cause of failures. Analysis is manual. Monitoring provides an assessment of nodes and containerized services.	No inbuilt tools, but provides support via third-party tools, such as ELK (logging) and Reimann (monitoring).	Basic inbuilt tools and also interfaces towards third-party tooling are required.
Scalability	Intra-cluster concept, it runs into problems for cluster federation. Envisions auto-scaling. 5000 node clusters, amounting to 150 000 Pods.	Scales better than K8s, since it provides a basis for deploying faster containers in large environments. Envisions auto-scaling. The claim is to be up to 5 times more scalable than K8s. 1000 node clusters with 30000 containers	Starting point is the scalability KPIs provided to DS.
Security	Pod security policies can provide access to a complete cluster. K8s model for service account tokens enables API servers to be reachable from every container by default.	Exchange of data between nodes is based on encrypted data and is shared with neighboring nodes with a gossip protocol. Encryption keys are stored only in the master nodes. Data is not encrypted in transit. Secrets (password, tokens, keys) are encrypted and stored as Raft data. PKI is used to enable nodes to communicate with each other securely	Encrypted exchange of data by default, based, e.g., on a social opportunistic protocol; overlay could be based, e.g., on DLT.
Workload migration	Manual configuration is required to provide adequate auto-scaling or load-balancing	Should perform adaptive load-balancing across a cluster.	Performs load balancing not just taking into consideration workload, but also taking into consideration other types of context and behavior inference, intra and inter-cluster.
Node support	Supports up to 5000 nodes	supports 2000+ nodes	expected to support at least 2000 nodes
Optimization target	1 large cluster	multiple small clusters	Overall optimization strategies, to balance both intra- and inter-cluster operations.
Networking	Flat overlay network interconnecting pods	Docker daemons interconnected via overlay networks. Overlay network driver.	Overlay should be content-oriented (data-driven), thus requiring a new degree of network abstraction.
Availability	High-availability, health checks directly performed in Pods	High availability, containers restarted if failures occur.	High availability and redundancy, supporting also mobility aspects, due to the integration of context-awareness.
Mobility support	Not supported	Not supported	Inbuilt support, due to better adaptation to the general conditions, internal and external to the system (context awareness).
Compatibility with other tools	Not compatible with any existing Docker CLI and Compose tools	some support for third-party tools	Should ensure adequate integration of varied third-party tooling.
Architecture	client-server model	Native clustering approach	Clustering approach
Adaptability	Setup configuration	Setup configuration	Integrates behavior inference (ML) to support a higher degree of automation during the setup of the application and lifetime management.

During a specific period of time (in a day, in a week) the worker relies on the device to collect information on machines. The worker then moves to another location, leaving the device on the specific premises. On the new premises, the worker will use another device, but the application must be transferred in advance.

While the worker is performing his tasks, the orchestrator predicts (based on learning of the worker mobility patterns) that the worker will likely move to a new location to continue his work. Therefore, it triggers a decision to offload the application to a device at the predicted location to allow work to continue without interruption. Part of the orchestration

implementation will be to ensure that the data required by the application is available locally. For example, if the application includes production monitoring, production plans, and related data should be prefetched to the device at the new location.

In this case, the regular mobility of the user represents a type of user behavior trigger, which can be fed into an orchestrator to best estimate when and where to offload the application workload and state, and to ensure that any required data are available.

B. SMART CITIES

Smart Cities benefit greatly from the use of AI services. Coupled with ubiquitous connectivity (LoRa/LoRaWAN, 5G, etc.), it allows cities to consider different sensing aspects to improve the overall city planning. An example can be Audio/Video processing at Edge servers co-located with cameras in parking lots. Today, the collected data is sent to the Cloud for further processing. Based on Edge computing, it is feasible to run specific AI/ML in co-location with cameras and process the data locally.

An adaptation based on specific context indicators brings benefits in the sense that the application running may adapt, in specific locations, the running ML model to best suit the context (e.g., peak number of vehicles in certain seasons) and the overall needs (e.g., energy reduction, or need to process data faster during a specific time of the day).

Here, a dynamic orchestrator would support the required adaptation. For instance, it would be feasible to perform system adaptation on the Edge nodes to run specific microservices based on service or situation needs (e.g., on a camera, count the number of electric vehicles around; assess abnormal situations based on noise).

C. FARMING

A set of UAVs is used in the context of crop life cycle monitoring. The UAVs carry a data analysis application, which supports local pre-processing of the collected IoT data, to circumvent the issue of intermittent connectivity, and to reduce energy consumption. The pre-processed datasets are timestamped and downloaded to an Edge node when connectivity is available. Thanks to the global view of data provided by Pathfinder, it is simple for a back-end application running in the cloud to gather the pre-processed data and complete the overall crop monitoring and planning. This can be done without implementing any application-specific mechanisms, aside from a naming convention including time-stamping.

D. MOBILITY

e-Mobility solutions are becoming increasingly decentralized and integrate different e-Mobility services (e.g., transports, energy monitoring) in an attempt to provide personalized recommendations to the user. Such recommendations require heavy data analysis, today performed in the Cloud. With the increase in the heterogeneity of possible e-Mobility services and with the integration of support for a more dynamic lifestyle, where commuting requires the use of different infrastructures, feedback to the user becomes more complex. The use of context-awareness and of a dynamic orchestrator facilitates the integration of new services, and the possibility to handle data locally. Here, the role of the orchestrator is to assist the data exchange across federated cluster environments, making it possible for the application to provide recommendations interactively in real-time.

As an example, consider an employee traveling to attend a meeting in another city. Based on user preferences and predicted traffic, a travel plan is created that includes a train to the destination city and then a taxi to the meeting location, which might be convenient in an unfamiliar city. If at the destination train station, local traffic is not as expected, the traveler may reroute his trip to use a local subway or tram system that bypasses unexpected road congestion. The orchestrator predicts the need to run the planning application on the Edge near the destination train station, with up-to-date data on local road and public transportation operations in order to interact with the traveler to make it easy to adapt the trip.

In this example, the orchestrator would rely on user preferences (e.g., preferred means of transportation), location (distance to a station), and dynamic information (traffic conditions or service interruptions) to ensure that the necessary information is collected to assist the user in making good, dynamic decisions for planning and adapting travel plans.

V. THE ROLE OF AI/ML IN CONTAINER ORCHESTRATION

A. KEY CHALLENGES

ML-based container orchestration technologies have been leveraged in cloud computing environments for various purposes, such as resource efficiency, load balancing, energy efficiency and *service level agreement (SLA)* assurance [26], [31]. One main challenge when making an offloading decision relates to the learning and adaptation capability considering both internal and external system variables. The load on the system resources, as well as the number of users and the data being consumed, change constantly, and both users and the system nodes may be mobile, thus adding to the dynamics of the overall system [32], [33]. Therefore, integrating intelligence into this process is both challenging and essential [34]. Managing computing resources and optimizing costs in multi-cluster Edge-Cloud environments are highly demanding tasks, as container adoption is growing using multiple container management platforms [25]. Whether on-premises, in public Cloud environments, or both, the operational overhead for container adoption is significant. As administrators cannot predict the computing resource demands of applications, they typically reserve more computing resources for an application workload and state than needed. Therefore, the integration of learning and prediction using AI/ML is recognized as an essential element of container orchestration [35].

B. FEATURES AND BENEFITS

Supporting elasticity and internal and external system dynamics are fundamental requirements in Edge-Cloud environments, and therefore, offering appropriate resource scaling is one of the most important features. Adopting AI/ML techniques in the orchestration process will ensure better adaptability, as it will increase overall operational efficiency and provide more flexibility by replacing manual configuration with digital intelligence, reducing the need for manual resource monitoring, tracking of data usage, calculating the optimal configurations, and changing the configurations accordingly [36]. These tasks are automated and become routine when they are completed using AI / ML techniques in the orchestration process. Common issues, such as over-provisioned computing resources and deployments with poorly selected number and size of pods, or under-provisioning of resources, e.g., for time-critical workloads, may be avoidable with ML. ML integration is also beneficial for helping to predict resource consumption appropriately in a way that can best adapt the system over time. Therefore, the use of AI/ML in orchestrators provides a learning mechanism for the patterns of use of the application resource and allows the prediction of resources down to the container level. While continuously generating recommendations, the system learns based on additional data. Overall, this can result in a reduction in spending while increasing the application service quality and delivering the necessary performance.

The interlock of AI/ML with the orchestration mechanism can provide precise offloading decisions based on the overall dynamic operation and state of a system, across time, and also taking into consideration the overall context of the system. Such a context can be based on different external and internal parameters, e.g., derived from network or application requirements; from data observability; from the user behavior as will be discussed later in subsection VII-A.

The optimization of the objectives and metrics of ML-based approaches for container orchestration has been investigated, and multiple methods have been extensively discussed in related literature. Figure 3 provides a taxonomy for the application of ML in different features of container orchestration [26]. This taxonomy addressed benefits ranging from resource efficiency, energy efficiency, and cost efficiency, for instance.

The proposed taxonomy details the use of ML in five specific orchestration categories: application architecture, infrastructure, optimization objectives, behavior modeling, and resource provisioning. In the context of the proposed taxonomy, the application architecture represents the behavior and internal structures of the containerized application components. Infrastructure indicates the environments or platforms where applications operate considers single Cloud, multi-Cloud, and Hybrid Cloud infrastructure patterns. Federated Clouds in this context fall into hybrid Clouds. Optimization objectives are the improvements that ML-based approaches attempt to achieve. Behavior modeling leverages ML models for pattern recognition and simulation of system and

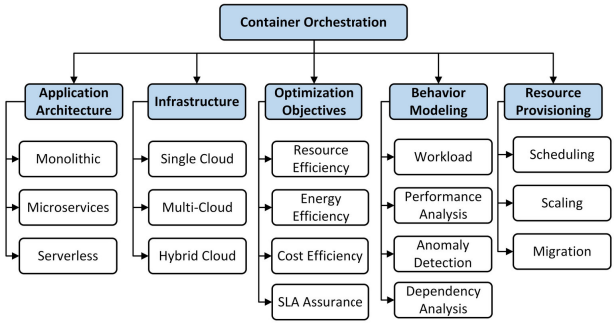

FIGURE 3. ML-based container orchestration taxonomy [26].

application behaviors, besides forecasting future tendencies according to collected data. Although resource provisioning prescribes the resource management policies of containerized applications at different phases of the container life cycle under various scenarios [26]. While relevant, the cited taxonomy isolates the use of ML across the different proposed categories, without considering, for instance, that several aspects are interlocked. For instance, optimization objectives provide examples of specific efficiency measures to consider, not considering the integration of other aspects, e.g., behavior modeling.

In our opinion, the integration of ML in orchestration serves the overall goal of improving adaptability and as such, requires a cross-category approach, where a central point is considering multi-objective optimization approaches that improve the overall system efficiency, adapting to different environments, without the need for additional manual intervention. For this purpose, Figure 4 provides a high-level perspective on the key aspects that are required in a dynamic orchestrator. As illustrated, we advocate the integration of context based on multiple parameters (data observability, application and network requirements, user behavior) that bring context-awareness to an adaptive ML-based layer, capable of behavior modeling and prediction. Such approach needs to be abstracted from underlying architectures, being able to be deployed on Cloud-based architectures, Edge-Cloud, or fully decentralized Edge Architectures. For this purpose, it is relevant to understand what kind of ML-based approach can be used to reach decentralization. These aspects are discussed next.

C. ML-BASED ARCHITECTURES FOR EDGE-CLOUD

The implementation of ML-based approaches generally follows three phases: data pre-processing, model training, and model testing. In data pre-processing, the learning task and the learning parameters shall be declared, while the datasets are loaded to perform the training and import the required libraries according to the learning task and datasets. Once the data is pre-processed, it is used to train the model. Afterward, the model is tested based on the achieved convergence accuracy and evaluation parameters. All these phases are conducted based on different ML architectures, of which

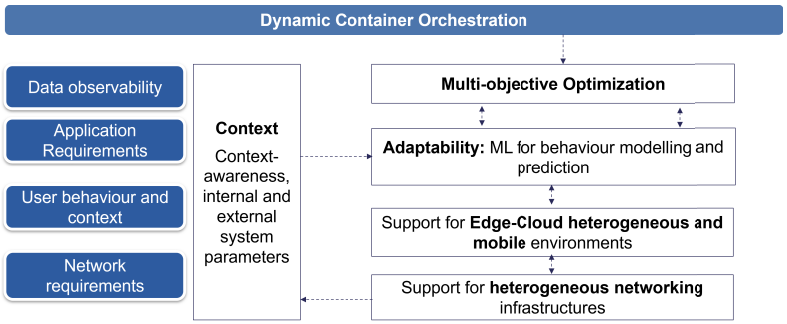

FIGURE 4. ML as a key aspect in dynamic container orchestration.

federated learning is an example [37]. This section aims at explaining main differences between existing types of architectures (centralized vs decentralized; distributed, federated and swarm learning). The section does not exhaustively describe different algorithms that can be used across Edge-Cloud, given that such aspects would require a deeper, survey-oriented analysis, and the focus of this work is on advocating a need for a cognitive and decentralized orchestration and to provide a first insight into the role of cross-layer context-awareness in this process. For this topic, we also direct the reader to related literature on the topic of AI approaches to the Edge-Cloud continuum [38], [39].

1) CENTRALIZED LEARNING

In centralized ML architectures, data training is performed in a central location, usually the Cloud. This often requires having an infrastructure capable of supporting heavy data transmission from a high number of data sources towards the Cloud. In general, centralized training is computationally efficient assuming a not so large number of sources, and adequate interconnections to the Cloud. However, centralized approaches imply, in particular when considering personal data, eventual security and confidentiality breaches, since data is stored in the cloud [37].

2) DISTRIBUTED LEARNING

The increased risks of moving large amounts of data to a centralized entity motivated the evolution of the distributed ML approach, where training, prediction, and inference are based on live streaming data. Distributed learning relies on a multi-node approach, where datasets are locally trained (per node). Distributed learning engages the server to distribute a pre-trained or generic ML model to the participating entities. Distributed learning is particularly beneficial when there are frequent and large updates of data, and, eventually, some devices are resource constrained - hence, distributed learning approaches become beneficial in Edge-Cloud environments [37].

3) FEDERATED LEARNING

Federated learning (FL) follows a hierarchical approach, where an algorithm training can consider datasets on different Edges, without having to transmit such datasets to the Cloud (only learning parameters are transmitted). Therefore, this technique reduces the amount of data transferred and minimizes the privacy concerns of the user's private data.

FL can be based on a centralized approach or on a decentralized approach. In a centralized approach, FL considers, for instance, a Cloud server to orchestrate the algorithm and ensure coordination of different learning parameters e.g., for different datasets on different edges. Hence, there is an initial selection of specific (edge) nodes to consider in the training and to aggregate the updates. This approach partially inherits the issues of centralized learning approaches, as the Cloud server can become a bottleneck. However, it prevents privacy issues. For the decentralized FL, the different involved entities (nodes) coordinate the global model to consider directly. In this case, the main issue may be related to the underlying infrastructure [37].

4) SWARM LEARNING

Swarm learning[6] is a decentralized approach to protecting privacy that is built on the principles of *Distributed Ledger Technology (DLT)*, thus supporting learning in a decentralized way near data sources. Raw data are kept local, while learning parameters are shared via a swarm network, and hence, models are built independently at the edge. Swarm Learning integrates security to support data sovereignty, confidentiality, via smart contracts involving on pre-authorized participants. Swarm learning therefore supports a dynamic onboarding of nodes, via smart contracts. Once a new node enters (via a smart contract), it obtains a model and performs local training until specific synchronization conditions are met. Then, the model parameters are exchanged via a swarm API, and merged to create an updated model with updated parameter settings, before a new training round is started. Warnat-Herresthal et al. provided an example of the application of Swarm Learning in the context of clinical disease classifiers, showing promising results in terms of decentralized learning, while maintaining data privacy [40].

5) SUMMARY

In the Internet and with the increasing integration of IoT across different vertical domains, sending large amounts of

[6]https://github.com/HewlettPackard/swarm-learning

data to the Cloud for training and inference introduces major issues when confronted with the existing decentralization trend in the Internet.

In this context, FL and its variants (centralized, decentralized, hierarchical) have been contributing to allow to decentralize the learning process across Edge-Cloud. Federated learning allows participant nodes to collaboratively train local models on their data without revealing sensitive information to a central Cloud. However, while it provides anonymity, it does not preserve privacy.

Swarm learning, on the other hand, seems to bring additional benefits in comparison to FL. The most visible is protecting privacy (privacy preservation). A not-so-visible benefit concerns the possibility of integrating more nodes (hence, more local training on different edges) and higher heterogeneity in terms of local model training.

To assist the evolution of container orchestrators, it is important to consider decentralized and privacy-preserving approaches, with Swarm learning being a relevant example to consider. Other variants of decentralized FL may bring advantages in this context, in particular approaches that consider aspects such as mobility [41], constrained devices, and synchronization [42].

Three specific challenges must be addressed by design, when considering future container orchestrators and keeping in mind the need to integrate a decentralized learning approach for orchestration: mobility of involved entities [43]; preservation of privacy, as attempted via swarm learning; and the capability to perform training involving constrained devices [44].

VI. THE NEED FOR DATA OBSERVABILITY IN IoT-EDGE-CLOUD COMPUTING

Data observability includes providing a view of many aspects of data from an organization, for example, including information describing where data is stored, how it is structured (*schema*), where it originated (*lineage*), how it is classified, how it is used, how it is protected - in order to allow greater value from data and, in general, better manage and protect the data. Data observability becomes increasingly important as IT environments are expanded to include Cloud resources, since like the IT resources themselves, the data becomes more distributed and more difficult to manage efficiently. Edge computing further increases the amount and distribution of data by adding compute and data domains that are closer to real-world data workflows across different domains such as Smart Cities, Agriculture, Manufacturing, Health.

A. MOTIVATION

Our objective is to improve the way data are used and managed in the Edge Cloud IoT and the internal processing and data domains of the company. We want to ensure that data can be found, that data pipelines work as intended to provide data for processing, and above all that data are protected and used in compliance with organization policies and applicable regulations. Improved data observability is essential for the orchestration of computation. By incorporating information describing data and the state of the networks and systems on which that data is processed, we will be able to significantly improve process orchestration.

Some of our specific objectives for improving data observability across compute and storage hierarchies include:

- **Knowing where data is stored and available for processing**: Data needs to be moved to where it will be processed, or processing moved to the data. Moving data may mean copying and possibly transforming complete data sets. Alternatively, data can be queried remotely to avoid copying. Rich data observability will make it possible to further optimize orchestration by accounting for the cost of accessing data.
- **The ability to exploit and manage data as things change**: In dynamic IoT Edge environments, events can occur that affect how data can be used or protected. The flow of data might be interrupted by a disruption in the systems where it is generated. A data pipeline might break due to changes in the data structure or a change in the IT environment. A new deployment of an application that processes source data to produce higher-value assets, such as machine learning models, may fail. The prevention or rapid detection of such situations is essential to minimize disruptions in the real-world systems that are monitored and managed.
- **Providing end-to-end observability**: The objective of Edge computing is to bring computation and data storage closer to where data is produced. Therefore, many Edge-computing scenarios will require the use of multiple Edge clusters to accomplish this. The administrative control of the domains may belong to different parts of a company or be provided by different service providers. Optimization of orchestration and data management requires visibility of data and the state of resources across these domains.
- **Compliance**: As data processing and storage becomes more distributed and data are moved or copied through the IOT-Edge-Cloud and on-premise domains, the risk of violating either company policies or regulations grows. Therefore, it is essential to monitor these environments to ensure that the data is stored and processed in compliance with applicable policies and laws.
- **Data Protection**: Similarly, in dynamic IoT-Edge-Cloud environments, data becomes more exposed to theft or tampering. In the case of sensitive data, special attention is required to ensure that the data is properly protected while at rest and when moved or copied. Processes that include vulnerabilities that expose data, for example, by encrypting data using weak or poorly configured cryptographic algorithms, might be deployed. All cryptographic components (algorithms, configuration, keys, certificates) must be adequate to achieve the required level of protection.
- **Flexibility to deal with diverse systems**: Expanding existing distributed IT environments to include Cloud

and Edge resources requires monitoring and controlling an increasingly wider range of systems. Therefore, the monitoring system must be easy to adapt to these new systems.

Orchestration across multiple domains can benefit from rich, cross-domain data observability. Orchestration needs to account for the cost of moving data or moving processing, therefore considering factors such as the size of the datasets, frequency of access, frequency of data updates, and network bandwidth and latency. The availability of data lineage information and metrics on the frequency of updates and freshness of copies of datasets may help with locating copies of data that enable better overall orchestration decisions. In general, making good orchestration decisions requires having an up-to-date view of the resources available across all domains. In particular, triggers need to be available to make it possible to adapt the system to changes that may disrupt running processes, thus warranting re-orchestration.

B. PATHFINDER DATA OBSERVABILITY

Pathfinder is a system developed by IBM Research to provide observability of data by automatically collecting, linking, and enriching metadata describing data, as well as the systems that process and store the data [45], [46]. Pathfinder collects metadata from a wide range of sources so that information that is normally siloed is available from a single source to simplify orchestration and data management tasks. The complete collection of linked and enriched metadata is called the **Enterprise Data Map (EDM)**.

Pathfinder is **event-oriented** - built on Apache Kafka - to support applications that need to be informed of changes in system state.

Pathfinder uses **connectors** to interface with the systems to collect metadata. For example, a connector to a database or file system can provide information on the tables or files that are available. Connectors to data catalogs can provide metadata on how data is classified, who owns the data, and other relevant information found in catalogs. Connectors to data pipelines provide information on data lineage and transformations that have been performed on source data to produce new datasets. The Pathfinder connector model ensures that the system can be extended to support new sources of metadata.

The independence of Pathfinder connectors from the core system simplifies the deployment and administration of the system in an environment with multiple administrative domains. The Pathfinder core system does not require any permissions to access any of the systems from which metadata is collected. The connectors collect and share metadata that can be shared with the core system. The only requirement is that the connectors and core system have connectivity. Connectors can be deployed and provided with credentials under control of the domain in which they are run. Thanks to this flexibility, Pathfinder can create an end-to-end view of data and other resources, including across multi-domain environments.

Pathfinder uses **enrichers** to add additional information to the EDM based on the collected metadata combined with other information and analysis. As a simple example, an enricher could be created to analyze the quality (relative to some desired metric) of a dataset, which is then added to the metadata and thus available to all applications that might process that data. A more complex enricher would be required to evaluate policies, making use of the information in the EDM to determine if data is being stored and processed in compliance with those policies. In this case, Pathfinder has a compliance role by implementing data security and privacy controls [47]. If metadata is available that describe known vulnerabilities to processes running in an IT environment, this view can be enriched by tagging data assets that are exposed due to those vulnerabilities.

Significant value in a data observability system comes from the fact that data is linked and enriched so that the collection of metadata (the Pathfinder EDM) provides more information and value than the sum of the individual pieces. For example, if the EDM includes the data classification learned from a data catalog and lineage information learned from a data pipeline to show where the data has been copied and how they have been transformed, a compliance enricher can be created. Since we view data compliance evaluation as one of the fundamental elements of Pathfinder, the system must have a data model that explicitly includes all the information required to implement this enricher.

If an application wants to include metadata that is only of interest within that application, it is sufficient that the connector for that application adds the information with an appropriate tag linked to the relevant element(s) in the EDM. This information cannot be interpreted by Pathfinder but can be used by all components of the application that understand it.

VII. DYNAMIC CONTAINER ORCHESTRATION FRAMEWORK

The previous sections focused on presenting container orchestration as it is today and existing tools that can provide better offloading support; AI/ML for the learning and system adaptation; Pathfinder as a relevant tool to provide data observability results.

In this section, we propose a conceptual framework to support dynamic container orchestration, as represented in Figure 5, which provides a high-level scheme for the functional blocks of the proposed dynamic container orchestration.

This framework could be implemented within an existing orchestrator or as a third-party application. The explanation provided in this section considers the case where such a framework would be implemented as a third-party application to K8s, thus interacting with the K8s scheduler. The different components are explained in the following subsections, after explaining what type of context should be considered to achieve a more elastic container orchestration.

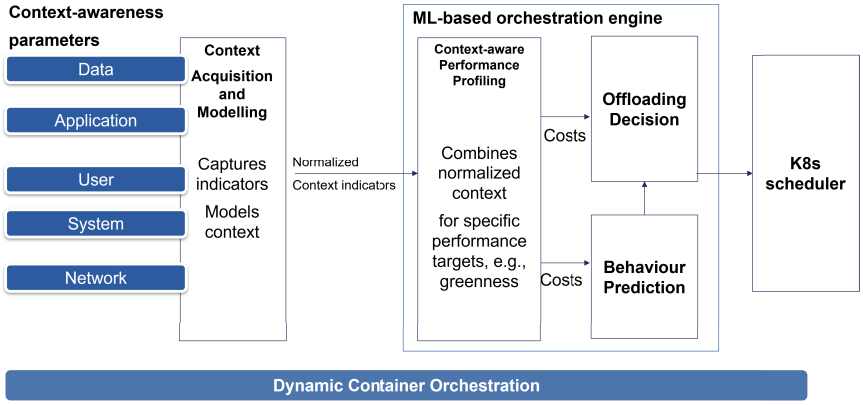

FIGURE 5. High-level block diagram of the proposed framework.

A. CONTEXT-AWARE OFFLOADING TRIGGERS

In current container orchestration solutions, the need to replicate resources is usually based on system requirements (e.g., CPU usage of a node reaches a specific level or energy level is perceived as high); application requirements (e.g., bounded latency); network policies. To further support dynamic Edge-Cloud environments, container orchestration needs to integrate context-awareness into the offloading process.

A first step towards achieving context-aware orchestration is the definition of a basic set of context-awareness parameter categories to consider in the offloading process, how to model context, and how such modeling can be integrated.

Our proposal is to consider the following categories of context-aware triggers as illustrated in Figure 6 and discussed next: i) system; ii) network; iii) application; iv) data observability; v) user behavior and preferences.

The acquisition of context parameters is handled via the context acquisition block. This requires a sensing module or a sensing interface that may be present on each worker node in an orchestration system and is passively fed with context. It can also be performed via a monitoring protocol that may require active probing instead of passive sensing. The Pathfinder system described in Section VI-B provides a general mechanism to generate some of the context-aware triggers. Pathfinder connectors interface with the systems distributed across the Edge-Cloud environment and gather information on their status. This information is then available in the Pathfinder EDM. For example, server utilization statistics can be gathered across all domains. If a particular server's utilization exceeds a defined threshold, an event can be generated to trigger re-orchestration. Similar triggers can be defined to ensure that the system reacts to network delays and problems with applications (e.g., if an application is no longer responsive, it can be restarted). Pathfinder enrichers also generate data compliance triggers based on how the data is classified, how it has been transformed, where it is stored, and the purpose for which it is being used. Note that if information cannot be determined automatically (for example, for processing), reports can be generated for offline investigation.

1) SYSTEM TRIGGERS

System triggers refer to internal parameters of a system, such as CPU or memory utilization, that can be engineered to improve the efficiency of the overall system. In current orchestrators, the overall Edge-Cloud infrastructure is modeled in accordance with the needs of an application, to adequately scale over time. For instance, an application may require more CPU capability than the capability existing at a specific node. The orchestrator system adds such a capability on the basis of available nodes. Common system parameters available in orchestrators are CPU, memory, and storage. Energy consumption can be integrated into an orchestrator scheduler, via third-party solutions.

2) NETWORK TRIGGERS

Regarding the networking level, the context surrounding the nodes in different container orchestration tools can help to better define opportunities for replication / transfer over time and space. Current orchestrators rely on a networking overlay and do not consider specific functional or non-functional networking requirements. A containerized application is often deployed on a collective set of nodes, which are interconnected via an IP network, public, or private. Moreover, there is frequent interaction between containers and between containers and the data they process.

Accordingly, each container should be accessible and discoverable thanks to the container networking mechanism.

The horizontal Pod autoscaling feature of K8s provides service availability and scalability by increasing the number of replicas/Pods. As a result, load-balancing between replicas of an application is necessary by distributing requests/demands equally to all pods in a K8s cluster. Taking into account that these requests might be forwarded to remote worker nodes, this approach can result in long delays, especially in Edge computing environments where the worker nodes are geographically dispersed [48].

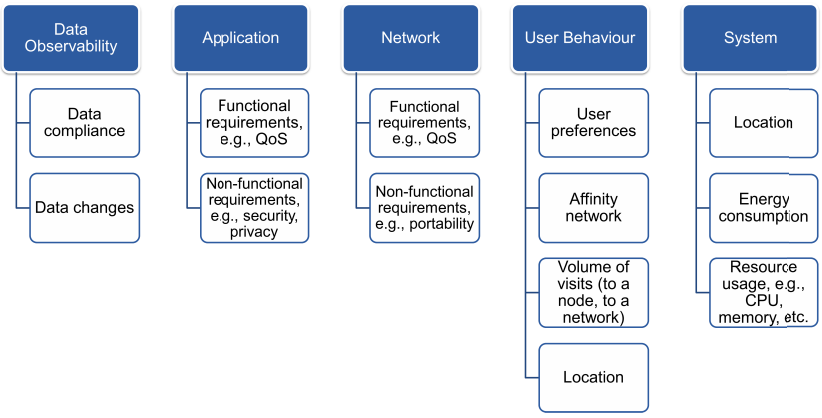

FIGURE 6. **Proposed context-aware categories and examples of specific parameters that can be sensed.**

Starting from the RTT, latency and the allocated bandwidth between the worker node and the master node, up to traffic load-balancing operations, varied QoS aspects from a node, path, and link perspective are examples of functional requirements that should be considered in the orchestration process. Examples of non-functional requirements are, for instance, mobility or portability.

Currently, and due to the fact that orchestrators rely on overlay networks, what is considered from a network perspective is *Service Level Agreements (SLAs)*. SLAs are not enough to provide an adequate adaptation of the underlying infrastructure. In mobile environments, where there are frequent IP changes handled by mobility solutions such as *Mobile IPv6 (MIPv6)* [49]. *Network Function Virtualization (NFV)* today provides the ability to integrate mobility management into orchestrators [50], thus bringing the ability to better adapt the overall orchestration of the system, taking into account computational and networking resources. Considering an approach similar to the one of MIPv6, it is possible, for instance, to handle changes of IP subnets across an old Pod and a new Pod.

3) APPLICATION TRIGGERS

Current orchestrators rely on application functional requirements to best model the scaling or load balancing of the system. However, support for time-sensitive applications is limited, as explained in Section II, where we have described the related literature that proposes changes to the K8 scheduler to better meet the needs of time-sensitive applications. In the context of industrial IoT, for example, critical applications often require a minimum and a bounded latency within one microsecond. Tight time synchronization among devices serving a specific application is often a challenge, due to time-aware queuing disciplines that support the exchange of critical traffic. These are examples of QoS parameters. Current orchestrators require a finer-grained support for QoS to cope with new challenges. Adding to this, nonfunctional requirements, such as a sensitiveness level of an application or even certification or compliance aspects, may prevent an application to run under specific conditions on a node.

Other requirements, such as application usage, roaming and location preferences, or even desired time to completion of a service, can be provided via a semantic abstraction of the application during setup.

4) DATA OBSERVABILITY TRIGGERS

Events to re-orchestrate application processing can also be triggered by a solution such as Pathfinder based on data-related changes. The most basic event of this type could be created on the basis of a new or updated data set. These datasets could be tagged in the EDM by the application or pipeline that creates them, or a Pathfinder enricher could create triggers automatically based on predefined conditions. Many different characteristics of the data set could be considered. For example, if a dataset is not updated within a specified time, or if a dataset grows to exceed a specified size, a trigger to be generated to schedule application to handle the condition.

Triggers can also be created based on data compliance issues. Regulations, such as the European),[7] and company policies can constrain, for example, where data can be processed and the purpose of that processing, and are therefore an important factor to be considered as part of orchestrating data processing.

In many cases, data protection issues should be evaluated when initial orchestration decisions are made, before data is transferred and processed. This is the only way to ensure compliance. In distributed Edge-Cloud environments, it will not always be possible to control all compliance issues a priori. For instance, data may be copied by a procFor example, data may be copied by a process that is not controlled (e.g., if that process is not documented) from an Edge to the Cloud for processing. To cover such cases, it is important to implement compliance monitoring to complement real-time controls.

[7]https://gdpr-info.eu/

This can be implemented using a data observability system like Pathfinder, which monitors the data and processing in each domain and uses enrichers to evaluate the compliance status. In some cases, a compliance violation could result in an orchestration trigger to move processing so that it is compliant. In other cases evaluation of a particular policy may not be possible (for example, if the purpose of processing by an application has not been documented), in which case a report should be generated to trigger manual processes to ensure that compliance is maintained or restored.

5) USER BEHAVIOR TRIGGERS

The generalized application of pervasive technologies, such as IoT, IIoT, and MCS, gave rise to the possibility of exploring user behavior across different domains, to improve system and networking behavior. A common indicator of user behavior is the location of the user, which is widely applied to improve the support of pervasive applications or the overall behavior of the system [51]. However, new technologies allow to explore and to integrate other facets of user and human behavior, in an attempt to benefit the overall orchestration of resources, in particular across mobile heterogeneous Edge-Cloud environments. An example of how user behavior can be applied to improve overall interconnection between users has been developed in the context of the H2020 UMOBILE project,[8] where a context-aware agent has been developed to capture and model user mobility to assist overall network operation and, in particular, to support social-aware opportunistic routing.[9]

Other forms of user behavior, including user preferences, have been considered, e.g., social interaction, group formation, and social/physical proximity [52] to improve overall system performance and to provide better support for task offloading across the Edge [53]. User mobility has also been applied in the context of improving energy consumption [54].

User behavior and preferences are therefore relevant to be considered in the context of improving container orchestration, in particular for mobile environments, as has been briefly explained in Section IV.

While there are a few starting points to model user behavior, it is relevant to understand which particular features can be considered, and how such context can be passed to the orchestrator scheduler, to improve the overall use of resources across Edge-Cloud.

6) SUMMARY, BASIC SET OF OFFLOADING TRIGGERS

In summary, a precise offloading decision depends on a set of operations and parameters that are required to be performed in an automated and dynamic way. These parameters are currently contingent on the triggers from the orchestrator system, the networking features, and the application requirements. However, it is feasible to further improve this by integrating both *external* and *internal* triggers to the orchestrator. For instance, taking data protection issues into account and integrating also human behavior into the orchestrator scheduler.

A proposed initial set of triggers based on the categories discussed is provided in Table 3, where the first column provides the type of trigger, system (**S**); application (**A**); network (**N**); data observability (**D**); user behavior (**U**). The second column identifies the parameter that shall be part of the set of offloading triggers, while the third column provides information on the current support in K8s. The fourth column explains the purpose of the integration of the parameter; the fifth explains units to consider, while the sixth column provides a description on how the parameters can be modeled.

The parameters taken into account by an orchestrator should not be worked in isolation, as they often depend on other different parameters that must be monitored by the orchestrator. For example, perceived QoS levels depend on other aspects, such as mobility or user location. By providing a multi-objective integration of the proposed parameters, it is our belief that the overall orchestration will become more fluid, and provide a better response to the needs of applications and the user.

B. CONTEXT ACQUISITION AND MODELING

The context acquisition integrates two subcomponents, illustrated in Figure 7.

First, it senses and gathers parameters, *context-aware offloading triggers*, that are described in Subsection VII-A. These parameters are generated based on changes in state of the system, network, and application, taking into consideration data observability as well as user behavior. The parameters are sensed or discovered (e.g., via a tool such as Pathfinder) as has been described. The parameters can be statically configured as happens today. The acquisition component collects and stores the different parameters in different multivariate time series. It may also perform local data aggregation per category, to reduce the resulting dataset.

The multivariate time-series datasets are then passed to the normalization and standardization component. Here, the datasets are first normalized (and, if required, standardized). This is required since each triggering type has a special unit, as described in Table 3.

Accordingly, context normalization avoids these problems by creating new values that maintain the general distribution and ratios in the dataset, while keeping values within a scale applied across all numeric columns used in the model.

The result is then passed on to the performance measurement component.

C. CONTEXT-AWARE PERFORMANCE PROFILING

The normalized context datasets are passed to the context-sensitive performance profiling block, illustrated in Figure 8.

This block performs a combination of the received data sets based of pre-configured heuristics that aim at providing a measure of performance, for a specific performance efficiency profile. For instance, assuming a user wants to

[8] https://umobile-project.eu

[9] https://umobile-project.eu/phocadownload/papers/wp-contextualmanager.pdf

TABLE 3. Initial proposed set of context-aware offloading triggers, and how they can be modelled to be integrated in K8s, as an example of an orchestrator.

Type	Parameter	K8s Support	Purpose	Modelling
S	CPU usage	Min-Max	Prevent computational exhaustion	Integrated into K8 as *container.resources*
S	Memory usage	Min-Max	Prevent storage exhaustion	Integrated into K8 as *container.resources*
S	Energy consumption	No	Optimize energy consumption	Rf. to Scaphandre for a modeling [10]
N	Path length (hops)	No, only network policies	Reduce latency	Interconnection between pods across multipath environments can consider a hop count distribution conditioned to Euclidean distance to estimate a best path to consider.
N	Network congestion	No	Improve resilience, support mobility	Based on jitter and packet loss modeling.
N	Available Bandwidth	Partial, limits can be set via network policies for a cluster	Reduce latency and improve resilience	Consider an adaptive approach based on passive probing to increase or decrease bandwidth.
A	Security	Partial, ACL, focus on authorization and access control only	Consider communication encryption to prevent security breaches	Integrate adaptive link-level security
A	QoS	Limited, based on specific user configuration	Integrate diverse application QoS levels, to allow for the support of real-time, time-sensitive, and bandwidth-intensive applications	consider the use of the DiffServ codepoint [55].
A	Geo-location	No	Take into consideration the application geolocation needs due to resilience or compliance	model the cluster in accordance with the geographical proximity of the micro-services.
D	Data New or Updated	Yes	Ensure that new or updated data are processed in a timely fashion	If a new data set is created or a data set is updated within the defined scope, schedule registered application to process the data set.
D	Data Freshness	No	Ensure that data is updated at minimal intervals	If a threshold is exceeded, schedule a registered application to handle the issue.
D	Data Compliance	No	Ensure adherence to regulations and company policies	If a policy violation is detected, schedule a registered application to handle the violation.
U	Geolocation	No	Replicate/offload workload and state in a way that is closer to the user, to improve latency, resilience, or even due to compliance	Model the cluster in accordance with the user geolocation (centrality measure based on the user location)
U	Volume of visits to an IP network	No	Improve resilience and latency based on user affinity to a network	Combine the frequency of visits with the duration of visits on a specific IP network (between).

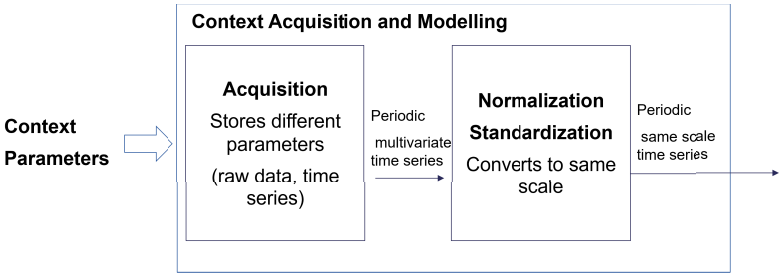

FIGURE 7. High-level representation of the context acquisition and modelling.

optimize the system for greeness, this block would select and combine weighted context datasets (e.g., hop count, energy consumption) in accordance with a specific function (e.g., product of hop count and energy consumption).

The combination of parameters is weighted, that is, the different categories of context parameters are weighted according to data preferences, or even according to learning over time, provided via feedback provided by the ML-based offloading decision engine.

Moreover, the combination of parameters takes into account feedback obtained from the ML-based offloading engine.

D. ML-BASED OFFLOADING ENGINE

The third component of the framework is the ML-based offloading engine, where different mathematical methods are applied to train and classify the optimized performance metrics' dataset, to assist the K8s in a decision to perform offloading.

In this block, ML is employed to provide behavior inference and also to perform prediction that can assist in understanding the impact that the proposed optimization may bring to the system.

[10]https://hubblo-org.github.io/scaphandre-documentation/tutorials/kubernetes.html

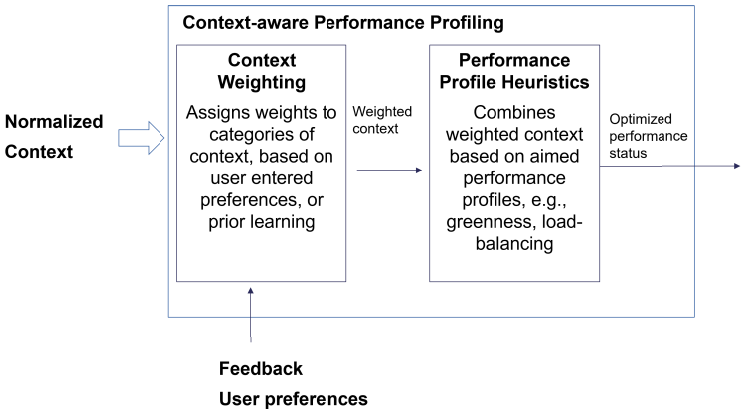

FIGURE 8. High-level representation of the context-aware performance profiling block.

This engine is therefore composed by a predictor component, and also by an offloading decision component. The predictor helps to estimate the impact that the combined context-aware optimization metric proposal may have on the system. The ML offloading decision engine learns about the change in demands derived from contextual information and proposes offloading configuration recommendations to the K8s scheduler.

E. CROSS-LAYER AND CONTEXT-AWARE ORCHESTRATION CHALLENGES

The capability to inject metadata and additional data into an orchestrator brings in the possibility to select an infrastructure that best adapts to the user expectations (QoE) and application requirements (QoS). In this context, there are a few challenges to be tackled, described in this subsection.

1) MOBILITY

Across a dynamic Edge-Cloud continuum, the overall infrastructure (data-network-computation) is dynamic. The network of computing nodes and connected devices is dynamic, and its state varies. Usually, a network overlay is established between pods, relying on the existing Internet routing. Hence, a first challenge concerns supporting mobility, and performing a placement that can provide lower latency and energy consumption. This implies offloading ''closer'' to the user/data sources. For this, context-awareness needs to consider aspects that derive from social proximity, and social mobility.

The first way to tackle this challenge is to move applications as close to the end-user location as possible. However, assuming that the end user has frequent movement, this will result in additional overhead in terms of signaling and also in terms of energy consumption [33]. Possible integration of mobility estimation as context awareness [56], to assist the scheduler in making a decision on whether or not to reschedule applications across the Edge-Cloud continuum. Mobile-Kube [33] proposes to consider as additional metrics energy consumption, resulting in an optimization of the overall infrastructure based on mobility and energy. Similar approaches, articulating the placement ''closer'' to the data source/end-user can be worked by considering additional context-aware metrics.

2) CONTEXT-AWARENESS INTEGRATION

One of the key challenges in container orchestrators such as Kubernetes is to be able to provide a cross-layer orchestration, thus allowing placement decisions to occur based on real-time resource demands that relate with the application and computational nodes; with the network; and also with the data. The current available schedulers provide partial support to these aspects, as has been explained in Section VII. When considering constrained devices and also intermittent connectivity/variable quality network links, this problem becomes more complex. A first challenge in this context is the definition of a subset of parameters that can provide a meaningful definition of a node, covering not just computational resources but also networking resources (e.g., egress and ingress bandwidth; centrality value), data freshness/compliance and energy consumption based not just on the energy consumed due to computational processes, but also due to networking processes. For examples in parameters, we refer the reader to the work under development by Sofia et al. in the context of the Horizon Europe CODECO project.[11] Specific parameters that provide an initial starting point are publicly available in the CODECO report D9, Annex I [57].

VIII. APPLICABILITY EXAMPLE: THE MoveK8s PROOF OF CONCEPT

To better illustrate how such a dynamic orchestration framework would work in practice, this section describes a proof-of-concept, MoveK8s,[12] which has been implemented in a Raspberry PI testbed in the fortiss Industrial IoT Lab, for a Manufacturing use-case. Movek8s has a Technology Readiness Level of 3 (TRL3), so it concerns *''Research that Proves the Feasibility of the Concept''*. The proof-of-concept aimed at understanding in practice how context-awareness in a simple form could be integrated into K8s and how

[11] https://he-codeco.eu
[12] https://git.fortiss.org/iiot_external/movek8s

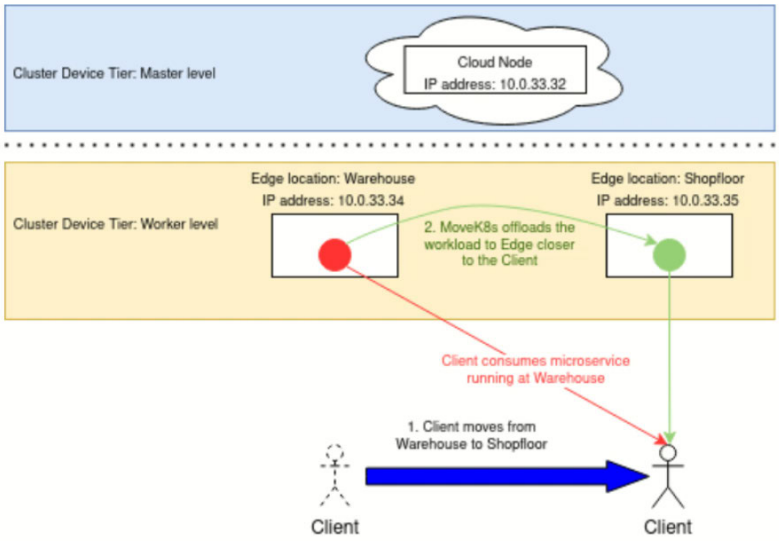

FIGURE 9. Movek8s topology and K8s architecture.

workload migration (and not replication) could be deployed. Aspects such as scalability, performance impact in the sense of resource utilization, and impact of several external (and internal) factors in the overall system are out of the scope of this perspective paper and expected to be addressed in a near future.

The underlying scenario for the proof of concept is represented in Figure 9, where three different nodes have been set in a single K3 cluster. Each node corresponded to a Raspberry Pi 4 Model B. The nodes have been labeled as Edge1, Edge2, Cloud. Then an Edge-based IIoT application for environmental monitoring in manufacturing environments, TSMatch,[13] has been set on the 2 Edge nodes. The initial location of the user is Edge 1 (IP: 10.0.33.34). The selected application integrates multiple micro-services, each running on a separate Docker container: (i) graph database, (ii) TSMatch Broker, and (iii) TSMatch Engine. Moreover, the end-user relies on an application that interacts with the TSMatch engine located on the Edge device, performing IoT service requests (e.g., monitor temperature on a room and send alert when temperature reaches a threshold).

From a K8s perspective, this deployment corresponds to a simple cluster, where each Edge node has been set as a worker node, and the Cloud has been set to run the K8s master node. Movek8s has been developed as a third-party Web-based application (Flask) co-located to the main node.

Movek8s performs a request to K8s to perform a change of workload based on location. For this, on this first proof-of-concept, the user relies on a QR code on each location. When the user moves from Edge 1 to Edge 2, there is currently a manual activation of the new location (via a QR code). The active location information is then sent to MoveK8s. A more efficient approach could be to predict the mobility of the user based on the pattern of visits to the new location, and to pass that information to the future Movek8s decision handling engine represented in Figure 5. Moreover, in the current proof-of-concept, Movek8s is placed in the main node. However, in future versions, we envision that Movek8s (or micro-services thereof) will also have to be installed on worker nodes as well. For example, the context acquisition and modeling functional block represented in Figure 5 should interface with each worker node.

Once the active location is received, Movek8s creates the application workload on the selected location and deletes the workload on the prior location (if it exists).

K3s handles, as usual, the interconnection of nodes within the cluster. Each worker node knows the IP address of the main node and the API port K3s and its K3s token. Movek8s gathers the user location provided and interacts with K3s to activate the TSMatch replica that is closest to the user-provided location. For the selected TSMatch Edge-based application, the Movek8s offloading process is as follows:

1) The three TSMatch containers belong to the same Pod, on Edge 1.
2) When a new location reaches Movek8s, it triggers the offloading of TSMatch to Edge 2.
3) The previous pod (Edge 1) is deleted, thus terminating the active TSMatch containers.
4) Movek8s then schedules the creation of a new pod containing the TSMatch containers, on Edge 2.

Policies defined to implement data security and privacy protection can restrict the orchestration process. In the previous example, if cameras are used in the warehouse to monitor and control a process, the images may include pictures of the people working there. The company may therefore define a policy that these images may only be processed in the local edge domain. Before implementing the offloading described above, the orchestrator should evaluate the applicable policies, and in this case decide to leave the processes

[13] https://git.fortiss.org/iiot_external/tsmatch

running on the warehouse if the workload is processing these images. The information required to recognize this condition is provided by the observability of the data. In this example, we assume that the privacy-protection policy has priority over the location trigger.

IX. SUMMARY AND FUTURE WORK

Next-generation IoT applications are expected to be deployed in a highly dynamic Edge-Cloud environment, involving different types of wireless and cellular infrastructure, mobile and limited nodes, and large-scale deployments. At present, the spread of different functions across IoT devices, Edge nodes, and the Cloud is achieved via virtualization. Containerized applications require flexible management, today provided by solutions such as K8s. However, such tools have some limitations in the context of decentralized environments. They handle the management of applications across Edge-Cloud in a static approach based on human intervention. This paper proposes an architecture for dynamic container orchestration based on ML and context awareness, integrating parameters that relate to application requirements, data requirements, system requirements, network requirements, and user behavior to support a more elastic orchestration of containerized applications. The components of the proposed framework have been explained, showing the relevance of context-awareness integration, the different types of its corresponding relevant parameters, and how they can be modeled and integrated into the orchestrator, enabling devised container orchestrators with enough elasticity, adaptation, and learning capability. The relevant role of data observability and trends towards decentralization have been advocated. Finally, the application of context-aware based orchestration has been developed in a testbed as a proof-of-concept to understand operational limitations derived from the integration of context. The presented work is the starting point for context-awareness integration in the context of the Horizon Europe CODECO project. The proposed categories of parameters are being used to further refine a cross-layer notion of context awareness that can be modeled and integrated into K8s as a representative example of a container orchestrator.

Future research should address the integration of context-awareness into the orchestrator scheduler(s), by considering approaches beyond the K8s filter and score (e.g., graph utilization maximization) that can meet application and user needs taking into consideration different layers of orchestration (network, computational, data, users, etc.).

REFERENCES

[1] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, and J. P. Jue, "All one needs to know about fog computing and related edge computing paradigms: A complete survey," *J. Syst. Archit.*, vol. 98, pp. 289–330, Sep. 2019.

[2] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang, "A survey on the edge computing for the Internet of Things," *IEEE Access*, vol. 6, pp. 6900–6919, 2018.

[3] A. Brogi, S. Forti, and A. Ibrahim, *Predictive Analysis to Support Fog Application Deployment*. Hoboken, NJ, USA: Wiley, 2019, ch. 9, pp. 191–221. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119525080.ch9

[4] J. Pan and J. McElhannon, "Future edge cloud and edge computing for Internet of Things applications," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 439–449, Feb. 2018.

[5] A. S. Thyagaturu, P. Shantharama, A. Nasrallah, and M. Reisslein, "Operating systems and hypervisors for network functions: A survey of enabling technologies and research studies," *IEEE Access*, vol. 10, pp. 79825–79873, 2022.

[6] K. M. M. Fathima and N. Santhiyakumari, "A survey on evolution of cloud technology and virtualization," in *Proc. 3rd Int. Conf. Intell. Commun. Technol. Virtual Mobile Netw. (ICICV)*, Feb. 2021, pp. 428–433.

[7] A. Randal, "The ideal versus the real: Revisiting the history of virtual machines and containers," *ACM Comput. Surv.*, vol. 53, no. 1, pp. 1–31, Feb. 2020, doi: 10.1145/3365199.

[8] A. M. Potdar, S. Kengond, and M. M. Mulla, "Performance evaluation of Docker container and virtual machine," *Proc. Comput. Sci.*, vol. 171, pp. 1419–1428, Jan. 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050920311315

[9] E. Casalicchio, "Container orchestration: A survey," in *Systems Modeling: Methodologies and Tools* (Springer Innovations in Communication and Computing), A. Puliafito and K. Trivedi, Eds. Cham, Switzerland: Springer, 2019, doi: 10.1007/978-3-319-92378-9_14.

[10] S. Wang, J. Xu, N. Zhang, and Y. Liu, "A survey on service migration in mobile edge computing," *IEEE Access*, vol. 6, pp. 23511–23528, 2018.

[11] B. Wang, C. Wang, W. Huang, Y. Song, and X. Qin, "A survey and taxonomy on task offloading for edge-cloud computing," *IEEE Access*, vol. 8, pp. 186080–186101, 2020.

[12] F. Liu, G. Tang, Y. Li, Z. Cai, X. Zhang, and T. Zhou, "A survey on edge computing systems and tools," *Proc. IEEE*, vol. 107, no. 8, pp. 1537–1562, Aug. 2019.

[13] S. Dubey and J. Meena, "Computation offloading techniques in mobile edge computing environment: A review," in *Proc. Int. Conf. Commun. Signal Process. (ICCSP)*, Jul. 2020, pp. 1217–1223.

[14] A. Malviya and R. K. Dwivedi, "A comparative analysis of container orchestration tools in cloud computing," in *Proc. 9th Int. Conf. Comput. Sustain. Global Develop. (INDIACom)*, Mar. 2022, pp. 698–703.

[15] R. C. Sofia, Ed., "A vision on smart, decentralised edge computing research directions," NGIoT, Europe, White Paper, 2001, doi: 10.5281/zenodo.5837299.

[16] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, "Towards a better understanding of context and context-awareness," in *Proc. Int. Symp. Handheld Ubiquitous Comput.* Berlin, Germany: Springer, 1999, pp. 304–307.

[17] W. Liu, X. Li, and D. Huang, "A survey on context awareness," in *Proc. Int. Conf. Comput. Sci. Service Syst. (CSSS)*, Jun. 2011, pp. 144–147.

[18] L. Bulej, T. Bureš, P. Hnetynka, and D. Khalyeyev, "Self-adaptive K8S cloud controller for time-sensitive applications," in *Proc. 47th Euromicro Conf. Softw. Eng. Adv. Appl. (SEAA)*, Sep. 2021, pp. 166–169.

[19] F. Rossi, V. Cardellini, F. Lo Presti, and M. Nardelli, "Geo-distributed efficient deployment of containers with kubernetes," *Comput. Commun.*, vol. 159, pp. 161–174, Jun. 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140366419317931

[20] S. Zhang, T. Wu, M. Pan, C. Zhang, and Y. Yu, "A-SARSA: A predictive container auto-scaling algorithm based on reinforcement learning," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Oct. 2020, pp. 489–497.

[21] M. C. Ogbuachi, C. Gore, A. Reale, P. Suskovics, and B. Kovács, "Context-aware K8S scheduler for real time distributed 5G edge computing applications," in *Proc. Int. Conf. Softw., Telecommun. Comput. Netw. (SoftCOM)*, Sep. 2019, pp. 1–6.

[22] K. Kaur, S. Garg, G. Kaddoum, S. H. Ahmed, and M. Atiquzzaman, "KEIDS: Kubernetes-based energy and interference driven scheduler for industrial IoT in edge-cloud ecosystem," *IEEE Internet Things J.*, vol. 7, no. 5, pp. 4228–4237, May 2020.

[23] D. M. A. D. Silva and R. C. Sofia, "A discussion on context-awareness to better support the IoT cloud/edge continuum," *IEEE Access*, vol. 8, pp. 193686–193694, 2020.

[24] S. Böhm and G. Wirtz, "Cloud-edge orchestration for smart cities: A review of kubernetes-based orchestration architectures," *EAI Endorsed Trans. Smart Cities*, vol. 6, no. 18, p. e2, May 2022.

[25] C. Carrión, "Kubernetes scheduling: Taxonomy, ongoing issues and challenges," *ACM Comput. Surv.*, vol. 55, no. 7, pp. 1–37, Dec. 2022, doi: 10.1145/3539606.

[26] Z. Zhong, M. Xu, M. A. Rodriguez, C. Xu, and R. Buyya, "Machine learning-based orchestration of containers: A taxonomy and future directions," *ACM Comput. Surv.*, vol. 54, no. 10s, pp. 1–35, Sep. 2022, doi: 10.1145/3510415.

[27] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *Proc. USENIX Conf. USENIX Annu. Tech. Conf.*, 2014, pp. 305–320.
[28] J. Nickoloff. (2016). *Evaluating Container Platforms at Scale*. [Online]. Available: https://medium.com/on-docker/evaluating-container-platforms-at-scale-5e7b44d93f2c
[29] Y. Pan, I. Chen, F. Brasileiro, G. Jayaputera, and R. Sinnott, "A performance comparison of cloud-based container orchestration tools," in *Proc. IEEE Int. Conf. Big Knowl. (ICBK)*, Nov. 2019, pp. 191–198.
[30] A. M. Beltre, P. Saha, M. Govindaraju, A. Younge, and R. E. Grant, "Enabling HPC workloads on cloud infrastructure using kubernetes container orchestration mechanisms," in *Proc. IEEE/ACM Int. Workshop Containers New Orchestration Paradigms Isolated Environments HPC (CANOPIE-HPC)*, Nov. 2019, pp. 11–20.
[31] N. Naydenov and S. Ruseva, "Combining container orchestration and machine learning in the cloud: A systematic mapping study," in *Proc. 21st Int. Symp. INFOTEH-JAHORINA (INFOTEH)*, Mar. 2022, pp. 1–6.
[32] S. V. Gogouvitis, H. Mueller, S. Premnadh, A. Seitz, and B. Bruegge, "Seamless computing in industrial systems using container orchestration," *Future Gener. Comput. Syst.*, vol. 109, pp. 678–688, Aug. 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X17330236
[33] S. Ghafouri, A. Karami, D. B. Bakhtiarvan, A. S. Bigdeli, S. S. Gill, and J. Doyle, "Mobile-kube: Mobility-aware and energy-efficient service orchestration on kubernetes edge servers," in *Proc. IEEE/ACM 15th Int. Conf. Utility Cloud Comput. (UCC)*, Dec. 2022, pp. 82–91.
[34] P. Gonzalez-Gil, A. Robles-Enciso, J. A. Martínez, and A. F. Skarmeta, "Architecture for orchestrating dynamic DNN-powered image processing tasks in edge and cloud devices," *IEEE Access*, vol. 9, pp. 107137–107148, 2021.
[35] W. Sun, J. Liu, and Y. Yue, "AI-enhanced offloading in edge computing: When machine learning meets industrial IoT," *IEEE Netw.*, vol. 33, no. 5, pp. 68–74, Sep. 2019.
[36] Z. Rejiba and J. Chamanara, "Custom scheduling in kubernetes: A survey on common problems and solution approaches," *ACM Comput. Surv.*, vol. 55, no. 7, pp. 1–37, Dec. 2022, doi: 10.1145/3544788.
[37] S. Abdulrahman, H. Tout, H. Ould-Slimane, A. Mourad, C. Talhi, and M. Guizani, "A survey on federated learning: The journey from centralized to distributed on-site learning and beyond," *IEEE Internet Things J.*, vol. 8, no. 7, pp. 5476–5497, Apr. 2021.
[38] M. M. John, H. H. Olsson, and J. Bosch, "AI on the edge: Architectural alternatives," in *Proc. 46th Euromicro Conf. Softw. Eng. Adv. Appl. (SEAA)*, Aug. 2020, pp. 21–28.
[39] Z. Zou, Y. Jin, P. Nevalainen, Y. Huan, J. Heikkonen, and T. Westerlund, "Edge and fog computing enabled AI for IoT–An overview," in *Proc. IEEE Int. Conf. Artif. Intell. Circuits Syst. (AICAS)*, Mar. 2019, pp. 51–56.
[40] S. Warnat-Herresthal, H. Schultze, K. L. Shastry, S. Manamohan, S. Mukherjee, V. Garg, R. Sarveswara, K. Händler, P. Pickkers, N. A. Aziz, and S. Ktena, "Swarm learning for decentralized and confidential clinical machine learning," *Nature*, vol. 594, no. 7862, pp. 265–270, 2021.
[41] Y. Liu, J. Nie, X. Li, S. H. Ahmed, W. Y. B. Lim, and C. Miao, "Federated learning in the sky: Aerial-ground air quality sensing framework with UAV swarms," *IEEE Internet Things J.*, vol. 8, no. 12, pp. 9827–9837, Jun. 2021.
[42] M. Chen, D. Gündüz, K. Huang, W. Saad, M. Bennis, A. V. Feljan, and H. V. Poor, "Distributed learning in wireless networks: Recent progress and future challenges," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 12, pp. 3579–3605, Dec. 2021.
[43] J. Kang, Z. Xiong, D. Niyato, Y. Zou, Y. Zhang, and M. Guizani, "Reliable federated learning for mobile networks," *IEEE Wireless Commun.*, vol. 27, no. 2, pp. 72–80, Apr. 2020.
[44] A. Imteaj, U. Thakker, S. Wang, J. Li, and M. H. Amini, "A survey on federated learning for resource-constrained IoT devices," *IEEE Internet Things J.*, vol. 9, no. 1, pp. 1–24, Jan. 2022.
[45] S. Rooney, L. Garcés-Erice, D. Bauer, and P. Urbanetz, "Pathfinder: Building the enterprise data map," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2021, pp. 1909–1919.
[46] D. Bauer, C. Giblin, L. Garcés-Erice, N. Pardon, S. Rooney, E. Toniato, and P. Urbanetz, "Revisiting data lakes: The metadata lake," in *Proc. 23rd Int. Middleware Conf. Ind. Track*, in Middleware Industrial Track. New York, NY, USA: Association for Computing Machinery, Nov. 2022, pp. 8–14, doi: 10.1145/3564695.3564773.
[47] (2020). *NIST SP 800-53 Rev. 5, Security and Privacy Controls for Information Systems and Organizations*. [Online]. Available: https://csrc.nist.gov/publications/detail/sp/800-53/rev-5/final
[48] Q.-M. Nguyen, L.-A. Phan, and T. Kim, "Load-balancing of kubernetes-based edge computing infrastructure using resource adaptive proxy," *Sensors*, vol. 22, no. 8, p. 2869, Apr. 2022. [Online]. Available: https://www.mdpi.com/1424-8220/22/8/2869
[49] C. E. Perkins, J. Arkko, and D. B. Johnson, *Mobility Support in IPv6*, document RFC 3775, Jun. 2004. [Online]. Available: https://www.rfc-editor.org/info/rfc3775
[50] Á. Leiter, D. Huszti, N. Galambosi, E. Lami, M. S. Salah, P. Kulics, and L. Bokor, "Cloud-native IP-based mobility management: A MIPv6 home agent standalone microservice design," in *Proc. 13th Int. Symp. Commun. Syst., Netw. Digit. Signal Process. (CSNDSP)*, Jul. 2022, pp. 252–257.
[51] C. Harris and V. Cahill, "Exploiting user behaviour for context-aware power management," in *Proc. IEEE Int. Conf. Wireless Mobile Comput., Netw. Commun.*, vol. 4, Aug. 2005, pp. 122–130.
[52] R. C. Sofia, L. Carvalho, and F. M. Pereira, *The Role of Smart Data in Inference of Human Behavior and Interaction*. London, U.K.: Chapman & Hall, 2019, pp. 191–214.
[53] F. Saeik, J. Violos, A. Leivadeas, M. Avgeris, D. Spatharakis, and D. Dechouniotis, "User association and behavioral characterization during task offloading at the edge," in *Proc. IEEE Int. Medit. Conf. Commun. Netw. (MeditCom)*, Sep. 2021, pp. 70–75.
[54] A. Przybylowski, S. Stelmak, and M. Suchanek, "Mobility behaviour in view of the impact of the COVID-19 pandemic—Public transport users in gdansk case study," *Sustainability*, vol. 13, no. 1, p. 364, Jan. 2021. [Online]. Available: https://www.mdpi.com/2071-1050/13/1/364
[55] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, *An Architecture for Differentiated Service*, document RFC 2475, USA, 1998.
[56] O. Aponte and R. C. Sofia, "Mobility management optimization via inference of roaming behavior," in *Proc. Int. Conf. Wireless Mobile Comput., Netw. Commun. (WiMob)*, Oct. 2019, pp. 71–76.
[57] R. C. Sofia, Ed., H. Mueller, J. Solomon, R. Touma, L. G. Erice, L. C. Murillo, D. Remon, A. Espinosa, J. Soldatos, N. Psaromanolakis, L. Mamathas, I. Kapetanidou, V. Tsaoussidis, J. Martrat, I. P. Mariscal, P. Urbanetz, D. Dykemann, V. Theodorou, S. Ferlin-Reiter, E. Paraskevoulakou, and P. Karamolegkos, "CODECO D9-technological guidelines, reference architecture, and initial open-source ecosystem design v1.0," Partners have Contributed to the Deliverable, Jul. 2023, doi: 10.5281/zenodo.8143860.

RUTE C. SOFIA (Senior Member, IEEE) is currently the Head of the Department of Industrial IoT, fortiss GmbH, Research Institute of the Free State of Bavaria. She is also an Invited Associate Professor with University Lusófona de Humanidades e Tecnologias and an Associate Researcher with the ISTAR, Instituto Universitório de Lisboa. She was the Co-Founder of the Portuguese COPELABS Research Unit and the COPELABS Scientific Director, from 2013 to 2017, where she was a Senior Researcher, from 2010 to 2019. She was also the Co-Founder of the COPELABS spin-off Senception Lda, from 2013 to 2019. Her research background has been developed in industry, such as Grupo Forum, Lisbon; Siemens AG, Nokia Networks, Munich; and academia, such as FCCN, Lisbon; INESC TEC, Porto; ULHT, Lisbon; Bundeswehr Universitót, Munich. She holds more than 80 peer-reviewed publications in the fields of research interests and nine patents. Her current research interests include network architectures and protocols, the IoT, edge computing, edge AI, in-network computing, and 6G. She is an ACM Europe Councilor and an ACM Senior Member.

She was an IEEE ComSoc N2Women Awards Co-Chair, from 2020 to 2021. She is the IEEE ComSoc WICE Industry Liaison Deputy. She leads the 6G CONASENSE Platform. She is an Associate Editor among several venues, such as IEEE Access and *IEEE Network*.

DOUG DYKEMAN received the Ph.D. degree in computer science from the University of Waterloo, Canada.

He manages the AI for Data Integration Research Group, IBM Research, Zürich, Switzerland. In the career, he has focused on a range of networking and system management topics, spanning the telco (initially at Northern Telecom) and enterprise (at IBM) worlds. His personal focus has covered research, product development, and standardization in this space. His current team is focused on using artificial intelligence (AI) and other technologies to simplify data management in large organizations. The IBM Pathfinder data observability project is focused on using metadata to build a complete map of data in an organization. He investigating how to use this enterprise data map to support a range of applications, including continuous compliance, data and AI workflow maintenance, and achieving agility in managing the use of cryptography.

PETER URBANETZ is currently a member of the AI for Data Integration Team, IBM Research, Zürich, Switzerland. His recent projects have focused on data management in large organizations. The team created the Cognitive Enterprise Data Platform (CEDP) for IBM's Chief Data Office to provide a state-of-the-art platform for analyzing Hadoop-based data. From the CEDP experience, the team recognized the constraints associated with copying data to a common data analytics platform, and therefore developed the idea of managing data based on decentralized data stores–the data remains with the organization that manages it–with centralized metadata to make it possible to simplify the job of managing and analyzing data while removing any practical scalability limits.

AKRAM GALAL (Member, IEEE) was a Researcher with the IIoT Research Department, fortiss, Munich, from August 2022 to January 2023. During the Ph.D. studies, he was a Visiting Researcher with the National Institute of Standards and Technology (NIST), USA, in 2019, and a Visiting Researcher with the Interuniversity Microelectronics Center (IMEC), Ghent University, Belgium, in 2020. He served as a Solution Design Consultant with Tawasul Telecom, a regional provider of information and communication technology solutions in Kuwait, from 2014 to 2017. He also served as an Enterprise Networks Engineer with Telecom Egypt, the dominant Internet service provider in Egypt, from 2011 to 2014. He has participated in several national/international research projects in collaboration with multiple partners from academia and industry and has been funded either by the EU, such as 5GROUTES and MARSAL, or by the Spanish Government, such as TRUE5G and 5GCity. His research interests include the IoT, the Internet of Nano-Things, edge computing, software-defined networking, network function virtualization, and AI/ML for data communication.

DUSHYANT ANIRUDHDHABHAI DAVE received the M.Sc. degree in informatics from the Technical University of Munich. From 2021 to 2022, he was a Student Assistant with the Department of IIoT Research, fortiss, where he was responsible for the development of the Movek8s demonstrator development and installation. He has industrial experience, having developed, during the master's studies, an internship with the TUM-IBM OpenPower Project, from 2020 to 2021. His research interests include software engineering, ML-driven communications and orchestration, and edge computing.