Autonomous Lifecyle Management for Resource-efficient Workload Orchestration for Green Edge Computing

Francesc Guim, Thijs Metsch, Hassnaa Moustafa, Timothy Verrall, **Intel Corporation**
David Carrera, Nicola Cadendelli, **NearbyComputing**
Jiang Chen, David Doria, Chadie Ghadie, **Lenovo**
Raul Gonzalez Prats, **Cellnex Telecom**

Abstract— **Edge computing is an important pillar for green computation by bringing the Cloud resources to the Edge, serving real-time applications, and reducing the computing and network resources required to transfer data for processing in the Cloud. 5G brings network densification and enables massive IoT and V2X applications, which triggers the need for edge computing to host network functions and user-facing services in a converged edge platform(s). Several edge computing deployments are being observed by ecosystem players (telco, ISVs, chip vendors, CSPs, … etc.) for IoT or V2X services, however, focusing on converged network functions and services. The point that is still in its early stages is the dynamic workload orchestration across the converged edge platforms running network functions and multi-tenant IoT services with different compute requirements and different Service Level Objectives (SLOs). This paper focuses on autonomous life cycle management for converged edge platform(s) to enable resource-efficient workload orchestration, contributing to the green goal. We present a solution for intelligent dynamic resources configuration on edge computing platforms hosting multi-tenant services while guaranteeing the SLO for each service and helping green communication goal. The presented solution has been deployed in a trial, and we present results on efficient resources configuration.**

I. INTRODUCTION

THE cloud in the past two decades allowed applications and data to be accessed from anywhere, following a centralized approach for ubiquitous services access and for efficient use of resources. However, the increase in networking and connectivity capabilities that we observe with 5G [1] and the continuous growth of IoT and V2X services, led to the generation of a considerable amount of upstream data to the Cloud applications as well as downstream data to end-users. Consequently, the paradigm of centralizing applications infrastructure at the Cloud for resources efficiency does not hold anymore as a considerable amount of network and communication resources become a prerequisite to provide a data pipe to the Cloud, which consumes compute resources. Besides, the nature of the new applications requires more and more real-time processing. All this led to the proliferation of edge computing, shifting from centralized to distributed and saving network and communication resources.

5G and edge computing offers a converged compute plus communication infrastructure for cloud-native applications for new services and software-defined network functions. This creates value across chip manufacturers and platform vendors, telco, CSPs, hyperscalers, SW vendors (ISVs), and System Integrators (SIs) who contribute to the creation of the edge computing HW and SW infrastructure [2].

Edge computing allows for ultra-low latency response times and enhanced bandwidth availability compared to conventional centralized computation models. Additionally, edge computing helps compliance to data use policies – for example data is constrained to remain within a certain location or when data must be processed locally because the cost of transport or transport time is prohibitive. Services that use the edge include but are not limited to traditional network functions, connected self-driving cars, video surveillance, IoT analytics, video encoding, video analytics, speech analytics or retail services, among others. The following are the most representative use cases and services [3, 4] for the converged edge computing platforms:

- Network Functions Virtualization (NFV) [5], such as vRAN and ORAN, where the required resources are mainly CPU, FPGAs, and smart NICs.
- Content Delivery Networks (CDN), where the required resources at the Edge are mainly storage, CPUs, GPUs, smart NIC, and network bandwidth.
- Smart Manufacturing [6, 7], such as automated defect detection, automated guided vehicles (AGV), factory remote control, where the required resources at the Edge are mainly CPUs, communication, FPGAs, accelerators.
- Video Analytics, such as monitoring and surveillance applications, where the required resources are CPUs, storage, and accelerators (e.g., GPUs and VPUs).
- AR/VR and Gaming that require images processing, transcoding, stitching, and annotations with overlaid images, where the resources required are CPUs, communications, and accelerators (e.g., GPUs and VPUs).
- Connected and Autonomous Vehicles [8, 9], such as V2X and V2V applications for safety, traffic information, interactive maps, and passengers' entertainment, where the required resources are CPUs,

communications, and accelerators (e.g., GPUs and VPUs).

- Retail, such as shelves monitoring, self-checkout, cash desk monitoring, where the required resources are CPUs, communication, and accelerators (e.g., GPUs and VPUs).
- Health and Medical Applications [10, 11], such as digital imaging, tele-health, and remote surgery, where the required resources are CPUs, real-time and time-sensitive communication, accelerators (e.g., GPUs and VPUs).
- Smart Cities and Government Applications, such as connected transportation, smart traffic lights, traffic management, smart intersections, where the required resources are CPUs, communication, and accelerators (e.g., GPUs and VPUs).
- Audio AI [12], such as speech recognition and natural language processing for automation, interactive experience, and chatbots, where the resources required are CPUs, FPGAs, and AI accelerators.

These different services have different requirements in terms of edge location (and hence form factors), performance (and therefore compute resources), QoS, SLOs, data storage and sharing policy, and security. This creates a hard need for intelligent life cycle management of the edge applications, where end-to-end orchestration is crucial to meet the applications requirements and SLOs while continuously adjusting the platform and system parameters to ensure efficient usage of computing resources. Intel platforms provide a unique and wide range of control knobs that enable orchestrators with the ability to dynamically adjust the QoS of the applications while providing excellent performance predictability in shared and constrained environments.

This paper focuses on autonomous lifecycle management for green edge computing platforms through dynamic and intelligent orchestration allowing efficient use of resources while guaranteeing Service Level Objectives (SLOs) across a set of diverse services. The rest of this paper is organized as follows: Section II discusses the related work on resources orchestration in edge computing and frames the context of the proposed work, Section III shares the services orchestration known models, and Section IV presents the solution we bring for autonomous life cycle management across the converged edge platforms. Based on a real deployment as described in Section V, we showcase results in Section VI, and we conclude the paper in Section VII.

II. Background and Related Work

To pave the way for MEC deployment with 5G infrastructure, several efforts focused on services orchestration. The European Telecommunications Standards Institute (ETSI) has standardized the reference architecture for NFV management and orchestration [13]. MEC services and NFV functions are considered in [14] through a common framework to help network functions and services convergence, and [15] adopts the ETSI NFV MANO framework and expands it to Fog Computing to enhance the orchestration decision making.

The centralized and distributed management schemes in Fog Computing are studied in [16], focusing on network usage and latency, which showed that distributed management performs better, especially when the number of sensors connected to the Fog node is low. Distributed resource allocation and orchestration is presented in [17, 18], seeking the optimal partitioning of shared resources between different applications running over a common edge infrastructure, through a service-defined approach to orchestrate cloud/edge services, allowing each application to define its own orchestration strategy with declarative statements.

The 3rd party MEC renting infrastructure is considered in [19], and a MEC broker is introduced to handle the access policies/privileges across the tenants and allocate resources in compliance with each tenant SLA. A fuzzy-logic based approach is introduced in [20] for workload orchestration across the edge computing infrastructure for processing tasks from mobile devices. Edge/Cloud interoperability is also considered in [21] by proposing a toolkit to interact with multiple Cloud and Edge platforms in real-time and provide rule-based engine to specify underutilization and overutilization, which helps switching-on/switching off resources. In addition, orchestration within CDN is considered in [22] introducing virtual CDN (vCDN) with SW defined features to split caching and streaming across the shared resources, which allows the maximization of the QoS and the optimization of the limited edge resources.

For IoT deployment scenarios, services orchestration for IoT mobile networks is introduced in [23] through dynamic hierarchal orchestration algorithms and blockchain and reinforcement learning to allow trusted resource sharing and automatic adjustment. Several efforts consider resource orchestration for IoT edge gateways considering event driven, publish-subscribe mechanisms. [24] presents a light-weight service lifecycle management for containers workload in industrial IoT, helping service deployment, updating, and terminating across IoT gateways in smart factories. In [25] an edge computing solution is presented for resource management between IoT gateways, considering the context from data streams and the resources on IoT gateways and connection topology.

Moreover, several orchestration frameworks exist within the open source community for cloud-native applications orchestration in the Cloud and at the Edge, such as OpenStack [26], Kubernetes [27], Mesos [28], and Akraino [29]. Expansion of the Kubernetes scheduler is presented in [30] for dynamic orchestration based on real-time resources status, where the scheduler uses status data from the cluster nodes to decide on workload orchestration based on a measure of the actual current load and the requirements of new containerized applications (a.k.a, PODs).

The existing efforts in research, standards, and open source frameworks consider workload orchestration from the following specific standpoints: (i) orchestration of NFV workload, (ii) orchestration of workload across IoT gateways or edge computing server through hierarchal load balancing across multiple edge platforms/IoT gateways, (iii) orchestration of workload through resources division across multiple types of services (e.g., content streaming and content caching in CDN), and (iv) orchestration of workload across the shared resources in edge platform(s) through holistic knowledge on the resources over utilization/underutilization. Existing solutions fall short to: (i) provide workload orchestration through real-time and dynamic knowledge on each edge platform utilization in a very fine-grained manner across all HW elements in the platform, and (ii) associate this real-time, dynamic, and fine-grained platform HW utilization knowledge with the Service Level Objective (SLO) of each workload to setup in real-time the right HW configuration for each workload. The solution we provide in this paper alleviates this limitation and satisfies (i) and (ii).

III. Services Orchestration Models

The diverse services enabled by edge computing platforms vary in their requirements in terms of:

(i) *Power, cooling and form-factor constraints*, which in turn dictate how many resources can be placed in a particular location and how ambient conditions may influence how those resources behave or even their availability (e.g., solar power-based deployment),

(ii) *Performance and QoS*, where data for an autonomous car may have higher priority than a temperature sensor in terms of the response time requirement. Depending on each application, the possible performance bottlenecks can be a combination, of computing resources, memory, storage, and network,

(iii) *Reliability and Resiliency*, where some input streams need to be acted upon and the traffic routed with mission-critical reliability, whereas some other input streams may be ok to tolerate an occasional failure. Again, this is going to depend on the application, and

(iv) *Security*, where data protection policies, service isolation and attestation vary per service and location.

To meet the above requirements adaptively for each service, we expand the *Orchestration* definition to be the management of the services workload placement and processing through dynamic and intelligent resources configuration to meet the services level agreement. We consider single-site orchestration as well as multi-site orchestration.

A. Single-Site Orchestration

In this case, services, resources & infrastructure are deployed within a single Point of Presence (PoP) or a close geo-proximity. As most deployments will be hierarchically managed, the orchestration & management stack needs to adhere to the higher-level components' requirements and policies in the stack.

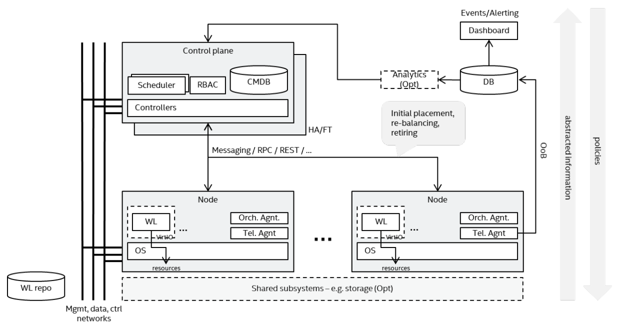


Fig. 1. Single-Site orchestration Example – showcasing multi-node deployment; Note that the control plane and its components can also run on a single node.

Fig 1. illustrates the flows and components of an orchestration and management stack in Single-Site deployment. It follows a classical Controller-Worker architecture and can be mapped to various orchestration solutions, such as, but not limited to, OpenStack [26], Kubernetes [27], and Mesos [28]. Furthermore, it is key that workloads define their Service Level Objectives (SLOs) and enable their continuous observability. This can be achieved through the right extensions to the manifest that are being used to deploy/provision the services. The provided SLOs can hence be monitored by the orchestration stack in place. The example in the Fig 1. shows a deployment with multiple compute nodes, it is possible to have single node configuration.

The footprint of the orchestration stack itself can vary while offering the same kind of interface. For example, Kubernetes K3s can be used on smaller devices, while providing similar functionalities. Controllers and schedulers are key components to translate SLOs into specific policies, while a set of agents and drivers enable the same.

To enable service performance and enable service assurance some key technologies are required: (i) Enablement of acceleration technologies and platform features within the orchestration in management stack, and (ii) Support for high precision orchestration and scheduling through (longer) running observations in a background flow, and a runtime driven foreground flow that deals with fine grained service request and re-balancing decisions.

B. Multi-Site Orchestration

Fig 2. Illustrates an example of multi-site orchestration. In this case, services, resources & infrastructure are deployed across multiple PoPs, where PoPs could eventually be mobile. A hierarchical, top-down management approach is required, where the following key requirements play a role when dealing with Multi-Site deployments: (i) Support for hierarchical management, (ii) Support for integration of various resource orchestration technologies/revisions, and (iii) Support for E2E orchestration for optimal placement/scheduling of service components considering resource constraints, data locality and SLO requirements.

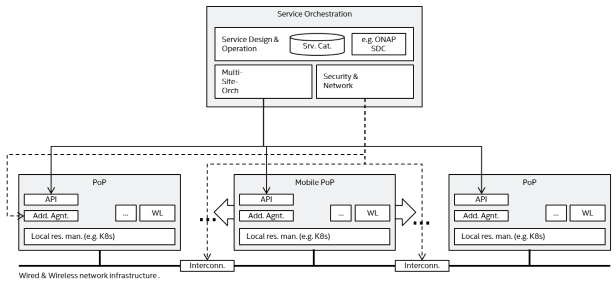


Fig. 2. Multi-Site orchestration & management flows – Akraino [29] reference solution example

From an E2E perspective and service assurance and enabling high QoS levels, it is key that the right policies and setting are translated from the top-down and broken down into individual policies and settings for the individual components. Overall monitoring and Service Level Agreement (SLA) management, Rating, Charging & Billing (RCB), and the possible remediation on SLA violation need to be handled in a top-down manner. This requires that the individual single-sites report the right (and enough information) to the higher layers.

IV. Autonomous Lifecyle Management

There is key motivation for autonomous lifecycle management for edge computing services. Today, independent software vendors (ISVs) and users are facing the challenge of developing and managing services at the edge. Edge infrastructure owners and current edge orchestration stacks assume that end-users will provide the resource requirements needed when a service needs to be launched at the Edge. Hence, a typical interface to launch an edge service would encompass: (i) The service itself that can be in different types of forms: binary, docker (descriptor, image or tag) or virtual machine, (ii) Potential list of data providers or users to be connected to the service, which can be a list of streams from specific cameras, IoT sensors using Messaging Queueing Telemetry Transport (MQTT) to provide data for instance from a factory, end-users streaming content from the service etc., (iii) Potential network latency and bandwidth requirements, and (iv) List of resources and resource requirements needed for the service, which typically includes a list of CPUs, accelerators, memory capacity and storage capacity, and other resources that can be potentially needed.

On the one hand, it is not straightforward for the ISVs or users to provide most of these requirements as they are associated with the use cases and are service domain specific. On the other hand, providing details on resources types/requirements is extremely complex for several reasons: (i) the current number of combinations of different type of processors (and the corresponding SKUS and different configurations) and other family of technologies (fabric, storage, accelerator etc.) can easily exceed thousands of potential edge appliances. This makes it impossible to understand how many resources are needed for a particular service to achieve the SLO in all the different possible combinations. (ii) Edge is a dynamic live system, which implies that applications may migrate from one location to another with other platform and resources. Hence, it's not feasible to expect the end user to know where the application will land or move as services may land or migrate to all the various edge platforms. This implies that loads on different edge platforms may vary and may require increasing resource assignment to critical workloads to keep with the SLO.

Another big challenge for the solution providers is that each edge infrastructure owner provides different ways to on-board applications. This implies that ISVs and other ecosystem players need to make a substantial amount of integration and interoperability effort every time they aim to show a new provider. This obviously does not scale and does not allow efficient maintenance and improvement for the applications.

The goal of the autonomous lifecycle management is to abstract as much as possible to the ISVs and other solution providers the complexity of the end to end edge architecture and mitigate the discussed challenges, through mainly: (i) Infrastructure abstraction, where services are not required to understand the available edge platforms and technologies up to a certain degree, but can be as optimized to utilize certain technologies when they are available (e.g., acceleration with AVX512, or a specific type of accelerators). (ii) Services description, where services provide to the infrastructure the required SLO to implement the use cases. Examples can be for instance that a video analytics workload can process N number of frames per second.

The autonomous lifecycle management that we present in this section provides real-time telemetry during the services lifetime, which the edge infrastructure can use to validate the SLO. The first level of orchestration occurs through automatic discovery of the resource requirement and the SLOs requirement for the on-boarded services. Then, the edge computing infrastructure with the edge stacks considers the real-time telemetry from the services to identify any SLO violation and decide whether to provide additional resource allocation to a particular service or migrate it.

A. *Solution Architecture*

The architecture's objective is to standardize services on-boarding at the edge and the interfaces between the services and the Edge platforms to perform the applications lifecycle management.

B. *Detailed Building Blocks in the Solution*

Fig. 3 illustrates the building blocks to address autonomous lifecycle management. The left part of the figure shows the discovery of the *service's golden configuration profile*. Its main objective is to discover, per each potential platform flavor (e.g., Xeon SP or Xeon SP + accelerator) the minimum needed resources for a particular service to satisfy the required SLO. The following subsection provides an example of how to implement such building blocks. These building blocks are part of the edge computing infrastructure that is abstracted to the end-users and ISVs.

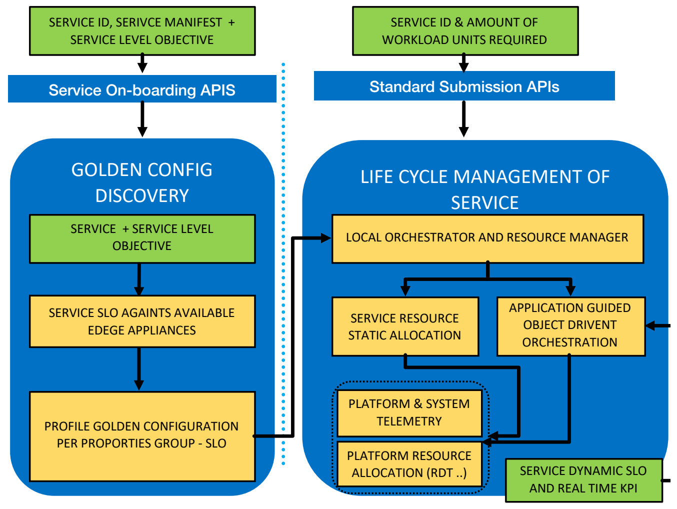

SERVICE ID, SERIVCE MANIFEST + SERVICE LEVEL OBJECTIVE
SERVICE ID & AMOUNT OF WORKLOAD UNITS REQUIRED
Service On-boarding APIS
Standard Submission APIs
GOLDEN CONFIG DISCOVERY
SERVICE + SERVICE LEVEL OBJECTIVE
SERVICE SLO AGAINTS AVAILABLE EDEGE APPLIANCES
PROFILE GOLDEN CONFIGURATION PER PROPORTIES GROUP - SLO
LIFE CYCLE MANAGEMENT OF SERVICE
LOCAL ORCHESTRATOR AND RESOURCE MANAGER
SERVICE RESOURCE STATIC ALLOCATION
APPLICATION GUIDED OBJECT DRIVENT ORCHESTRATION
PLATFORM & SYSTEM TELEMETRY
PLATFORM RESOURCE ALLOCATION (RDT ..)
SERVICE DYNAMIC SLO AND REAL TIME KPI

Fig. 3. Building Blocks – Case of Local Edge

The right part of Fig. 3 shows the actual orchestration building blocks and basic flows. The orchestration schemes use the *golden configurations* to decide resource selection and edge platforms selection depending on the desired amount of workload units (e.g., the need to process ten cameras performing anomaly detection). The lifecycle management of the service performed by the orchestration architecture is based on the application real-time KPIs provided by service.

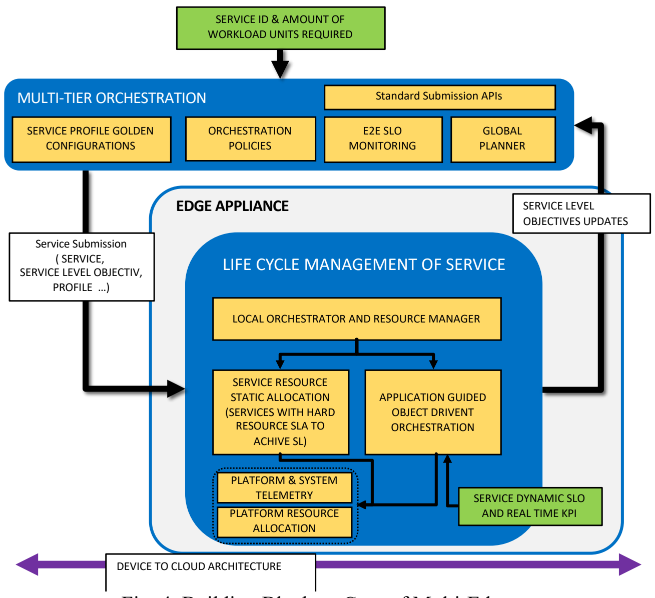

SERVICE ID & AMOUNT OF WORKLOAD UNITS REQUIRED
MULTI-TIER ORCHESTRATION
Standard Submission APIs
SERVICE PROFILE GOLDEN CONFIGURATIONS
ORCHESTRATION POLICIES
E2E SLO MONITORING
GLOBAL PLANNER
EDGE APPLIANCE
SERVICE LEVEL OBJECTIVES UPDATES
Service Submission (SERVICE, SERVICE LEVEL OBJECTIV, PROFILE ...)
LIFE CYCLE MANAGEMENT OF SERVICE
LOCAL ORCHESTRATOR AND RESOURCE MANAGER
SERVICE RESOURCE STATIC ALLOCATION (SERVICES WITH HARD RESOURCE SLA TO ACHIVE SL)
APPLICATION GUIDED OBJECT DRIVENT ORCHESTRATION
PLATFORM & SYSTEM TELEMETRY
PLATFORM RESOURCE ALLOCATION
SERVICE DYNAMIC SLO AND REAL TIME KPI
DEVICE TO CLOUD ARCHITECTURE

Fig. 4. Building Blocks – Case of Multi-Edge

Fig. 3 shows the architecture for a local deployment. For instance, it could be in a single edge appliance associated with an on-premise or private-5G type of deployment model (e.g., surveillance with a single edge platform performing the stream processing). However, in most edge deployments, the expectation is to have multi-tier edge appliances, where services can move from one location to other locations depending on the application performance or changes in the infrastructure (e.g., reduced network load). In this case, as shown in Fig. 4, there will be an interplay between a local edge orchestrator and a multi-tier edge orchestrator. For instance, if local orchestrator cannot make the service meet the required SLO it can work with the multi-tier orchestrator to migrate the service into another location.

C. *Autonomous Lifecycle Management – Full View*

As shown in Fig. 5, service life-cycle flow would take place as follows: (i) The orchestration entity running on that appliance is required to instantiate a specific service with a particular amount (which could be provided for instance with a set of data streams to connect). (ii) Based on the golden configuration and the available resources, the orchestrator will select the right service configuration and platform resources to satisfy the SLO associated with the service and based on the required quantity. For instance, if 10 streams at 1080 are provided to the service, and the service has associated 0.5 cores and 200 MB/s of memory bandwidth per stream to keep the SLO, the orchestrator may select 5 cores and allocates 2 GBS to those cores as starting point. (iii) Once the resources are selected, the orchestration logic will instantiate the service (configuring the data streams properly). (iv) Once the service is instantiated and starts reporting the real time application KPIs, the orchestration logic is responsible to monitor that the SLO is not broken. (v) If the SLO is broken, the orchestration logic may decide to allocate more resource or notify that the SLO cannot be achieved with the current load. There are multiple ways to identify what resources cause a bottleneck and need to be allocated (e.g., using Top-down Micro-architecture Analysis Method "TMAM" metrics or finger printing techniques). Our reference implementation provides a ML- based technique.

In many cases, the architecture will be composed of multiple edge appliances managed by an end-to-end orchestration, managing the end to end architecture at a higher level. Fig 4. shows an expansion of the previous scenario. In this case, the golden configurations may be hosted and managed by the higher orchestration entity and provide the resources required by the services and the SLO to the lower orchestration entity.

V. Implemented Architecture

This section provides a description of a deployment scenario that has been implemented based on the autonomous lifecycle management solution and architecture described in Section IV.

A. *Cellnex Green Edge Architecture*

The Cellnex Mobility Lab in Castellolí, near Barcelona (Spain), focuses on developing vehicular use cases. The lab results from the digital transformation of Circuit Parcmotor Castellolí, which has been converted into an innovative technology center that supports experimental living-labs for smart mobility and connected/autonomous vehicles.

The Mobility Lab develops 5G-based sustainable, connected, and autonomous mobility solutions for vehicles, traffic management, and road infrastructure. This represents an innovative area to test and develop new technological solutions and services advancing connectivity, especially in rural environments.

The racetrack has been equipped with several self-sustaining sites to support the Vehicle-to-Everything (V2X) wireless network that provides coverage to the whole circuit, allowing connectivity between vehicles, vehicles to infrastructure connectivity (high-definition cameras for monitoring vehicles on the track, environment sensors, IoT,…) for monitoring vehicles on the track and on-board units for transmitting telemetry, voice and video data.

The self-sustaining Green Edge sites are powered by renewable energy: in this case, the communication towers are equipped with wind turbines and solar panels and extended batteries to store the electricity generated on-site (no grid connection). The towers are interconnected using radio links, and therefore, there is no need for power cables nor fiber links between them.

One of the big challenges for this type of deployments is managing power consumption while keeping the right service level objective for the edge services running on the appliance and the service level objective that infrastructure stacks (such as vRAN) need. The architectural approach that we present addresses the previous challenge by reducing the gap between service and infrastructure flows.

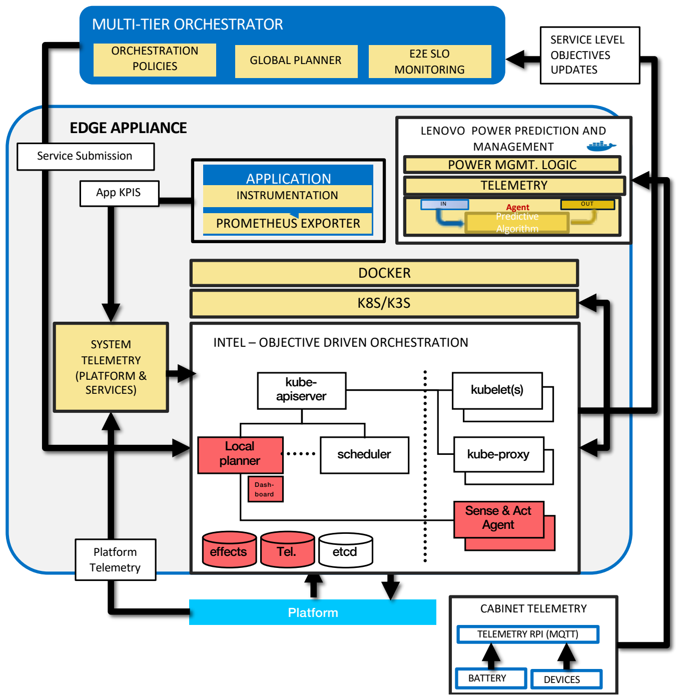


Fig. 5. Overview of the deployment scenario

B. *Deployed Scenario Description*

Fig. 5 illustrates the deployment scenario, which comprehends three main parts: (i) Global end to end multi-orchestrator that performs the edge appliance allocation when a service needs to be instantiated. Once the service is instantiated, it may receive a request from the lower orchestration entity running on the selected appliance if the SLO associated to the service cannot be met. (ii) Lower level orchestration stack comprising of Kubernetes using Docker and Prometheus application expansions that allow for reporting run-time telemetry corresponding to the SLOs defined at the submission time. (iii) The lower-level objective driven orchestration extension to the Kubernetes control plane tracks the service KPIs as reported, against SLO and potentially applies resource enforcement. (iv) The smart power management agent is responsible for predicting and managing the power at the system level to make solar-based edge appliances last.

C. *Multi-tier orchestration*

The global end to end multi-tier orchestrator is Nearby One [31]. A solution composed of two main elements:

- *The Nearby Orchestration Platform*: the main component of the solution, which runs in a central location and performs all the tasks related to the orchestration of applications and edge infrastructure.
- *The Nearby Blocks*: are distributed components that encapsulate logic and code for different application-specific functionalities.

The solution natively integrates with third party Virtual Infrastructure Managers (VIM), like OpenStack or VMware, and extensively supports the integration of Docker containers into both VIMs (containers in VMs) and bare metal servers. In this case, the applications are shipped as Docker containers and are deployed on emulated edge servers (VMs) that are treated as bare metal servers. Nearby One is also capable of provisioning and monitoring the infrastructure if needed, what is particularly relevant for low-end deployments in the ultra-far Edge, where no resources are available to run large VIMs.

Edge Applications, as deployed by the Nearby One solution, are encapsulated into Nearby Blocks to extend the ETSI MEC standard's capabilities. Application ecosystems require inter-application communication and accurate placement decisions, and they usually require advanced tuning of their execution platform. Each Nearby Block contains the application logic (containers or VMs provided by a third-party vendor), and they are encapsulated with a set of auxiliary components that provide the means for the application to be effectively managed, including:

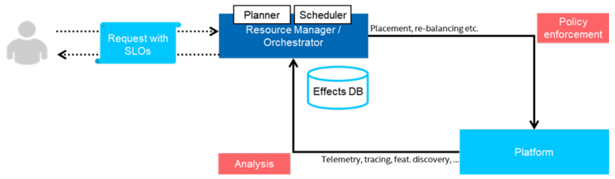
- Application performance KPIs for continuous SLA assessment (not only network-centric metrics).
- Application health/status KPIs for continuous lifecycle monitoring and management.
- Application capabilities dynamically exposed at deployment time (e.g., services exposed by the application for external consumption, like an output video feed, or a third-party management dashboard).
- Correlation of application KPIs and platform (processor, accelerators, memory, storage) for more effective MEC platform management.

D. *Local object driven orchestration*

We've made the following extensions to the Kubernetes control plane to be able to manage the objectives of a service/workload. These extensions include the following additional components (marked in red in Fig. 9) to the Kubernetes control plane:

- A (set of) telemetry/monitoring databases/sinks, where the platform's information and/or the service are stored. We use InfluxDB [32] instances from the service mesh Linkerd [33] and a standalone instance for storing information for our reference implementation.
- A database that stores the effects of orchestration actions / resource allocations have on the workload's objectives. In the initial reference implementation, we use MongoDB [34].
- A planner component, which can reason over the effect orchestration activities / resource allocation have on the objectives in place. It manipulates/changes the configurations of the workloads in place.
- A set of agents running on the nodes in the cluster, which can instrument the infrastructure and change resource allocation settings accordingly. In this reference implementation OPNFV Barometer's [35] collectd is used for telemetry, while a set of scripts is being used to manipulated resctrl [36] filesystem's schemata that are otherwise can be under control by CRI-RM [37].

Fig. 9 shows the high-level flows as also defined in [38]. In a foreground flow the user requests the deployment of a workload in combination with the associated objectives. The objectives are defined in terms of latency, availability, and throughput, which can be accompanied by violations budgets defining how often / how long an objective can be "violated". This is achieved by leveraging Kubernetes Custom Resource Definitions [39]. In a continuous running background flow, the planner observes the current state of the workload instance's (For example Kubernetes PODs as part of a ReplicaSet/Deployment) objectives and matches it against the desired state. This follows the default Kubernetes operator design pattern. The state is defined by the objectives associated with the workload instances, and the planning algorithms determine if any mitigation actions needed to be done (For example to minimize resource usage or optimize the configuration to better match desired state). The plan that is being derived can incorporate preferences from the resource owner in the form of tunable cost/utility functions. Possible actions include – but are not limited to – scaling out, removing "unreliable" PODs, tuning the associated Memory bandwidth, switching profiles, or changing of resource assignments. The resulting plan is executed upon by the K8s control plane – and if required the scheduler will place PODs according to the updated configurations. While the planner is in charge of translating the SLOs into actionable plans and defining what/how to set up the systems, it is the scheduler's responsibility to determine when/where to place the workloads according to the provided information in the deployment configuration.


Request with SLOs
Planner
Scheduler
Resource Manager / Orchestrator
Placement, re-balancing etc.
Policy enforcement
Effects DB
Platform
Analysis
Telemetry, tracing, feat. discovery, ...
Fig. 9. Foreground and Background Flow

Furthermore, in the background flow the effect's database is continuously being updated. Models that are either hand-tuned or machine-learned describe the effect actions have on the objectives of the workload. These models are being re-trained as new data from the telemetry systems becomes available. This enables tracking for workload changes, infrastructure changes and their behaviors over time. This also assures that various workload's characteristics will lead to a different set of useful actions. For example, a cache sensitive workload will react to cache tuning, while a micro service might benefit more from scaling out actions.

E. Power management

The power management solution is Lenovo Smart Power Agent. The Agent is composed of four main elements:

- *The input module*: It is responsible for collecting telemetry from the platform and the services of sensors. Metric data will be converted into suitable format and stored for offline analysis.
- *The predict module*: It uses the pre-trained ML model to make predictions with metric data from the input module.
- *The output module*: It provides interfaces for external users (e.g., Prometheus, ODO).
- *The control module*: It can set power limit automatically by BMC function or other knobs (e.g., RDT, SST).

The Agent is deployed on the nodes in the cluster and integrated with the orchestration platform. In this deployment scenario, the far-edge sites are solar energy-based. Each site was supported by renewable energy only and installed with three PV panels for power generation and one battery for energy storage.

To help each site's infrastructure last longer, the Lenovo Smart Power Agent collects telemetry from the sensors and uses this to predicts some key metrics. The prediction results include future available energy, remaining battery time and suggested power consumption. The output of the Agent can be used by the local planner to make decisions – e.g. if the suggested power capping can be set given the objectives that currently need to be fulfilled.

VI. Results

In the following sections we will showcase the obtained results for workloads management in a power constraint green edge. A key to managing the workload instances and their associated SLOs/objectives, is understanding how the workload behaves under various platforms loads . Rightsizing the workloads to match the desired objectives and load allows for reduction of resource allocations. Trading off the resource allocations in contrast to current load and desired SLO allow for more efficient management of the power budget. For example, lowering the resource allocations, or objectives – and slightly penalizing the workloads' performance – might allow for longer lasting batteries. Therefore, in the following subsection we show the results from analyzing the workloads and their performance, ways to automatically derive the profiles and their usage for orchestration decision.

A. *Definition of Services Golden Configuration Profile*

The minimum amount of resources defines a *golden configuration* for a particular service (e.g., memory, compute, acceleration etc.) needed for a specific service to achieve a certain SLO for a particular set of technology features. A group of technology features define a profile. This is not limited to specific SKUs or system configuration but considers a fundamental set of features that may dictate the performance and resource requirements for a service implementation. As an example, a particular service may have three golden configurations for three type of profiles: (1) CPU based execution with AVX512 as a required feature; (2) CPU based execution with FPGA as a required feature; and (3) CPU based execution without acceleration.

We also consider that one service may have different combinations of implementations (e.g., video analytics using different levels of precision) and inputs (e.g., stream resolution 1080 or 4K). Thus, a particular service may have multiple golden configurations mapped to multiple groups of features. This information allows the decision on where a specific service execution needs to land.

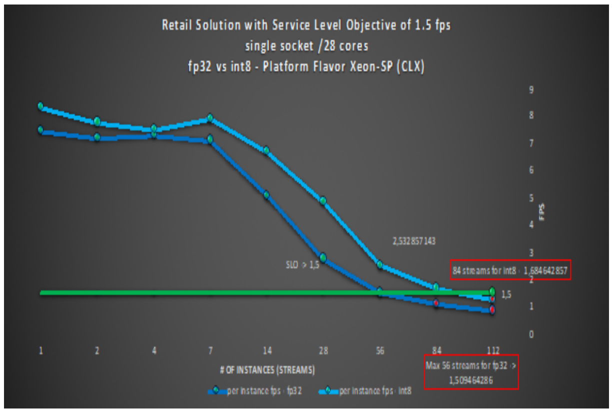


Fig. 10. Golden Configuration Identification for 2 Types of Streams

Fig 10. provides an example of a golden configuration identification. The figure shows how the frames per second per processed stream by a particular service (performing video analytics for retail) decrease as more streams are provided by the service. Two implementations are considered (INT8 and FP32). The SLO requires to perform 1.5 frames per second per stream. As observed this platform flavor (composed of Cascade lake platinum cores and NVME drives) can process up to 56 and 84 streams for the FP32 and INT8 implementation, respectively. After this point, adding more streams implies that the SLO cannot be met. The critical aspect of this exploration is dynamically identifying the minimum amount of resources required to achieve the given SLO.

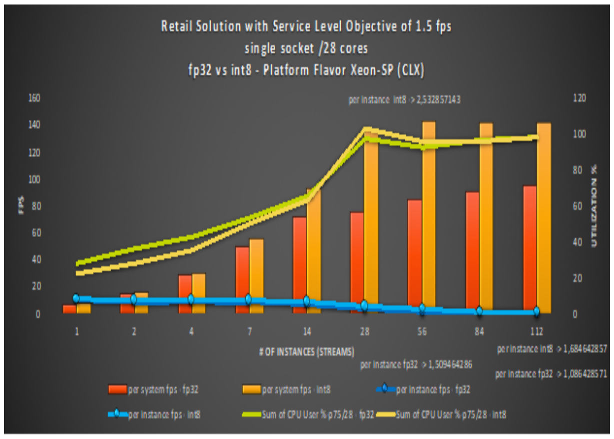


Fig. 11. Resource requirements associated to a specific golden configuration

Fig 11. provides an example of CPU utilization increase with the increase in the number of streams. The required CPU per stream will be decided function of the maximum amount of streams density that still satisfies the SLO. Hence, per the previous analysis, we select the point of 56 and 84 instances (per FP32 and INT8) and normalize the CPU utilization to those values: 28/56 cores/per stream for FP32 and 28/84 cores per stream for INT8. A similar process needs to be applied for the other resources (e.g., memory capacity, memory bandwidth etc.)

B. *Intelligent Discovery of Services Golden Configuration*

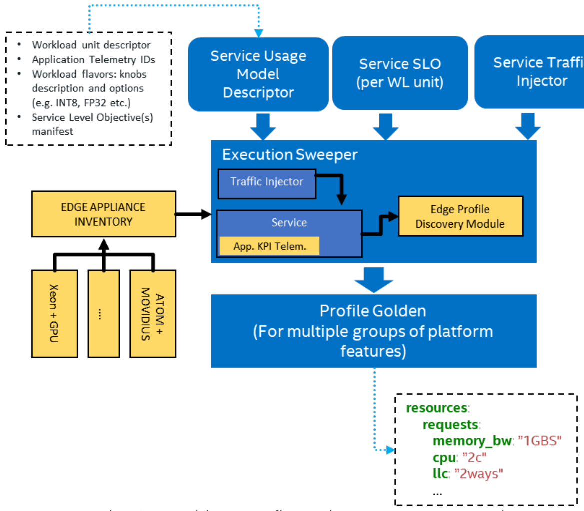
There are multiple ways to discover golden configurations. This can range from simple approaches that consist of a simple sweep of the application execution increasing the service demand to more sophisticated examples that may include ML techniques such as Random Forrest.

Fig. 12 shows the architecture of the Execution Sweeper. The inputs provided to the sweeper include:

- SLO definition
- Service usage model descriptor that includes elements such as docker image and knobs to be evaluated. A typical configuration may consist of two knobs: stream resolution (720,1048 & 4K) and neural network resolution (INT8, FP32)
- An injector module (that needs to be provided by the ISV) to generate traffic to the service. This injector must be capable of adjusting the amount of traffic (workload units) to the service
- List of platform flavors that need to be swept

The sweeper will create a golden configuration for each base line golden configuration per each profile. The execution sweeper, given all the previous inputs, will keep increasing workload units to the service until the SLO is not achieved anymore for a particular flavor, then establishes the golden configuration.

The current architecture implementation uses the sweeper to identify the resource requirements for different potential deployment options when services are on-boarded to the end to end edge deployment. Afterwards, multi-tier orchestration uses this information to select the right edge appliance based on the available profiles and the required estimated service instantiation resources. Future versions of the intelligent discovery will include federated learning to characterize and tune the golden configurations when services get deployed in the infrastructure and their requirements are better understood.



Workload unit descriptor
Application Telemetry IDs
Workload flavors: knobs description and options (e.g. INT8, FP32 etc.)
Service Level Objective(s) manifest
Service Usage Model Descriptor
Service SLO (per WL unit)
Service Traffic Injector
Execution Sweeper
Traffic Injector
Service
App. KPI Telem.
Edge Profile Discovery Module
EDGE APPLIANCE INVENTORY
Xeon + GPU
...
ATOM + MOVIDIUS
Profile Golden (For multiple groups of platform features)
resources:
requests:
memory_bw: "1GBS"
cpu: "2c"
llc: "2ways"
...

Fig. 12. Golden Configuration Sweeper Example

C. Orchestration decisions

Fig. 13 shows the KPI of a service instance in the system. In particular, the P50, P95 and P99 latencies quantiles are shown.

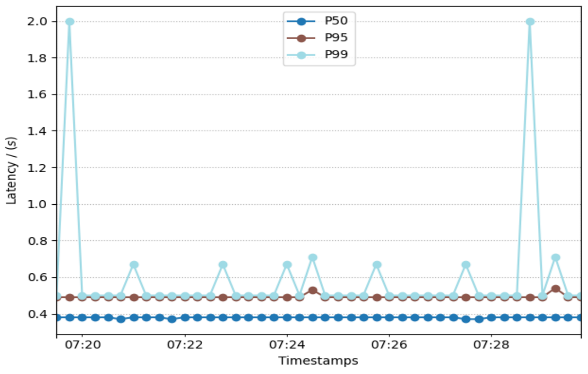


Fig. 13. Latencies values as reported by the services

Fig. 14 shows the effect of selecting a certain type of CPU over another on the latency of a deployed function.

This information is key to be able to understand the behaviour of the workload in the current context. Information on how latencies are effected by certain resource configurations – and hence golden configurations (discussed in Sectiob VI.A) – can for example be used to charterize workloads. The workload characteristics in turn can be stored in the effects database – as described earlier – and used by the local planner to make decisions.

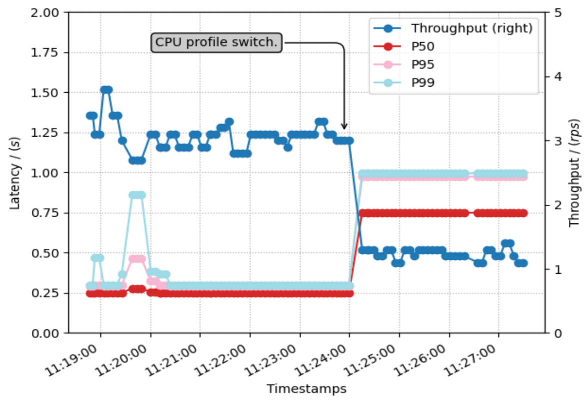


Fig. 14. Time series data showing the effect of selecting a certain CPU profile on P50, P95, P99 latencies – the planner uses this information to make the most efficient allocation (In this example a CPU with a lower TPU is selected, that can meet the requirements and is hence selected at around 11:24:00)

Similar to [40] and [41], we used a Bayesian Optimization (BO) algorithm – as part of archiecture described in the previous sectionB above- to determine the right amount of absolute CPU units and memory capacity to associate to a container instance. The CPU units are an absolute measure in Kubernetes, and hence an allocation is very specific to a certain platform, which a service owner might not know its details. Hence, the usage of a BO algorithm automatically determines the optimal resource allocation without the need to test all the possible options for resource allocation. The planner uses the information on the most efficient resource allocation and injects into the deployment description as CPU requests and limits (an example is shown in the following code listing for a video analytics stack component).

```
resources:
  requests:
    memory: "372Mi"
    cpu: "700m"
  limits:
    memory: "372Mi"
    cpu: "700m"
```

Rightsizing resource allocations for the workload is especially crucial for scheduling and admission control in a resource constrained edge. This avoids over- or under-provisioning of resources by the tenants. The planner can also trigger the workload migration and ensure the most efficient resource allocation. For example in Fig. 14 a different CPU resource profile is picked, which leads to an increase in recorded latencies, but can still fulfill the workload's objectives. The local planner also takes into account the exposed metrics from Lenovo's power Agent, considering the remaining battery time and the suggested power capping. If the remaining battery time falls below a threshold, the local planner adjusts the resource allocations (e.g., through reducing power or try to evict a workload instance from the pole). The suggested power capping is constrasted to the required power capping by individual workload instances to meet their objectives. Lower power envelopes normally mean reduced performance, wherethe planner trades off the required power capping in context of the desired objectives of the workload instances.

VII. Conclusion

With 5G and the emergence of new IoT services and the need for AI processing capabilities near the data sources, edge computing is moving from standards and concepts to true deployment. Edge computing is promising not only for reducing services latency, but also for helping green networking and communication by reducing network resources that would be needed if the services are always hosted in the Cloud. In this paper, we focus on a key aspect in edge computing, which is resources efficiency through autonomous lifecycle management for green edge computing platforms. We present the first of its kind architecture and solution that defines a golden configuration for each workload type to guarantee efficient use of resources while achieving the services SLOs. We present an example of a true deployment scenario and demonstrated some obtained data. In a later state we will incorporate further workload characterization techniques and investigate additional deployment scenarios with additional services.

References

[1] "View on 5G Architecture," 5G PPP White Paper, ver03, June 2019.
[2] D. Alusha, "Cloud-Edge Deployments in 5G Networks," ABI research report, June 4, 2020.
[3] A. Joshi, A. Kaul, "Artificial Intelligence Hardware and Software Infrastructure: Compute, Networking, Storage, and Cloud Infrastructure Driven by AI Deployments", Tractica Research Report, 2019.
[4] M. Saadi, D. Mavrakis, "5G and AI: the foundation for the next societal and business leap," ABI research report, April 2020.
[5] "Cloud Native Network Functions - Design, Architecture and Technology Landscape," Metaswitch White Paper, Nov. 2019.
[6] L. J. Su, "Machine Vision for Industrial Applications," ABI research report, December 2019. Smart manufacturing
[7] R. Whitton, "Mobile Robotics and Autonomous Material Handling for Logistics and Warehousing," ABI research report, November 2019.
[8] D. Sabella, H. Moustafa, et al. "Toward fully connected vehicles: Edge computing for advanced automotive communications," 5GAA White Paper, October 2017.
[9] H. Moustafa, E. Schooler, J. McCarthy, "Reverse CDN in Fog Computing: The life-cycle of video data in connected autonomous vehicles," IEEE Fog World Conference, 2017.
[10] L. Gergs, "5G in Healthcare", ABI research report, July 2020.
[11] H. Moustafa, E. Schooler, G. Shen, S. Kamath, "Remote Monitoring and Medical Devices Control in eHealth," IEEE WiMob, Workshop on eHealth Pervasive Wireless Applications and Services, 2016.
[12] "Transforming the Network Edge Enables New breakthrough Time Speech analytics," Intel-Verbio White Paper, December 2018.
[13]"Networks Functions Virtualization (NFV) Management and Orchestration," ETSI GS NFV-MAN, 2014.
[14] T. Doan-Van et al., "Programmable First: Automated Orchestration between MEC and NFV Platforms," IEEE Consumers Communication and Networking Conference (CCNC), 2019.
[15] S. Dalmini, M. Ventura, and T. Magedanz, "Design of an Autonomous Management and Orchestration for Fog Computing," IEEE International Conference on Intelligent and Innovative Computing Applications (ICONIC),2018.
[16] S. Dalmini, and M. Ventura, "Resource Management in Fog Computing: Review," IEEE International Conference on Advances in Big Data, Computing and Data Communication Systems (icABCD), 2019.
[17] G. Castellano, F. Esposito, and F. Risso, "A Distributed Orchestration Algorithm for Edge Computing Resources with Guarantees," IEEE INFOCOM, 2019.
[18] G. Castellano, F. Esposito, and F. Risso, "A Service-Defined Approach for Orchestration of Heterogeneous Applications in Cloud/Edge Platforms," IEEE Transactions on Network and Service management, Vol. 16 No. 4, December 2019.
[19] L. Zanzi, F. Giust, and V. Sciancalepore, "M2EC: A Multi-tenant Resource Orchestration in Multi-access Edge Computing Systems," IEEE Wireless Communications and Networking Conference (WCNC), 2018.
[20] C. Sonmez, A. Ozgovde, and C. Ersoy, "Fuzzy Workload Orchestration for Edge Computing," IEEE Transactions on Network and Service management, Vol.16, No. 2, June 2019.
[21] C. Anglano, M. Canonico, and M. Guazzone, "EasyCloud: a Rule based Toolkit for Multiplatform Cloud/Edge Service Management," IEEE 5th international Conference for Fog and Mobile Computing (FMEC),2020.
[22] J. Aires et al., "Phased-vCDN Orchestration for flexible and efficient usage of 5G edge infrastructures," IEEE conference on Network Function Virtualization and Software Defined Networks(NFV-SDN), 2019.
[23] S. Guo et al., "Trusted Cloud-Edge Network Resource Management: DRL-Driven Service Function Chain Orchestration for IoT," IEEE Internet of Things Journal, Vol. 7, No. 7, July 2020.
[24] H. Jo, J. Ha, and M. Jeong, "Light-Weight Service Lifecycle Management For Edge Devices In I-IoT Domain," IEEE International Conference on Information and Communication Technology Convergence (ICTC),2018.
[25]S. Pe, M. Radovanovi, and M. Ivanovic, "An MQTT-based Resource Management Framework for Edge Computing Systems," IEEE International Conference on INnovations in Intelligent SysTems and Applications (INISTA), 2020.
[26] OpenStack, https://www.openstack.org/software/
[27] Kubernetes, https://kubernetes.io/
[28] Mesos, https://mesos.apache.org
[29] Akraino Edge Stack API, https://www.lfedge.org/wp-content/uploads/2020/06/Akraino_Whitepaper.pdf
[30] M. C. Ogbauchi et al., "Context-Aware K8s Scheduler for Real-time Distributed 5G Edge Computing Applications," IEEE International Conference on Software, Telecommunications and Computer Networks (SoftCOM),2019.
[31] NearbyComputing, https://www.nearbycomputing.com/
[32] InfluxDB, https://www.influxdata.com/
[33] Linkerd Service Mesh, https://linkerd.io/
[34] MongoDB, https://www.mongodb.com/
[35] OPNFV Barometer Containers, https://wiki.opnfv.org/display/fastpath/Barometer+Containers
[36] User Interface for Resource Control feature, https://www.kernel.org/doc/html/latest/x86/resctrl.html
[37] CRI-RM, https://github.com/intel/cri-resource-manager
[38] T. Metsch, O. Ibidunmoye, V. Bayon-Molino, J. Butler, F. Hernández-Rodriguez, and E. Elmroth. "Apex Lake: A Framework for Enabling Smart Orchestration", In Proceedings of the Industrial Track of the 16th International Middleware Conference (Middleware Industry '15), 2015.
[39] Kubernetes – Customer Resources, https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/.
[40] Omid Alipourfard et al., "Cherrypick: adaptively unearthing the best cloud configurations for big data analytics", USENIX Conference on Networked Systems Design and Implementation (NSDI'17).
[41] Bin Li et al., "RLDRM: Closed Loop Dynamic Cache Allocation with Deep Reinforcement Learning for Network Function Virtualization", NetSoft 2020.