Intent-based Management for the Distributed Computing Continuum

Andrea Morichetta†, Nikolaus Spring*, Philipp Raith†, Schahram Dustdar†
Distributed Systems Group, TU Wien
† surname@dsg.tuwien.ac.at, * e11908527@student.tuwien.ac.at

2023 IEEE International Conference on Service-Oriented System Engineering (SOSE) | 979-8-3503-2239-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/SOSE58276.2023.00035

***Abstract*—Managing digital and connected systems has become increasingly challenging in the past decade due to their scale and complexity. A new perspective is required to manage these systems, considering the infrastructure and components from edge to cloud, i.e., in the distributed computing continuum. Serverless computing offers improved scalability and cost efficiency, but balancing and coordinating serverless systems remain complex. Intent-based systems, popular in networking, can provide a solution by translating stakeholder inputs into actions that meet Service Level Objectives (SLOs). Their application in the computing continuum can be highly beneficial, but it has yet to be deeply explored.**

To bridge this gap, we propose a methodology for deploying an intent-based system for the computing continuum. We implement an architectural framework leveraging the serverless paradigm. Furthermore, we focus on defining and implementing the main components for translating the management requirements into actions executed by serverless functions inspired by a three-layer model. Through a Proof of Concept (PoC) deployed in Amazon's AWS cloud and detailed simulations, we showcase how such an approach can resolve conflicts in a complex system, i.e., balancing efficiency and availability. Our work aims to contribute to effectively managing the computing continuum and highlight the potential of intent-based systems in this domain. The experiments' results show our framework's ability to make appropriate scaling decisions, fulfilling both objectives.

I. Introduction

In the last decade, the scale of digital and connected systems has reached the point where direct management is unfeasible. Global platforms like Google, Amazon, and Meta highlight the diversity and pervasiveness of infrastructure components, showing the complexity of their management [8], [53]. However, if we aim to handle these systems' complexity effectively, we must take a perspective where we consider their elements as part of a continuous entity. We call this entity the computing continuum, i.e., the ensemble of resources that spawn from the edge to the cloud. In the frame of the computing continuum, we have to work to develop efficient and autonomous management methods and strategies. Contextually, serverless computing, or Function as a Service (FaaS), has emerged as a promising paradigm for deploying complex systems applications. It offers various benefits [5] such as improved scalability and cost efficiency. In contrast to other cloud models, such as Platform as a Service, customers do not need to manage any infrastructure. Instead, they solely need to package their code as functions and upload it to the platform. In turn, the platform adapts the function deployments according to the workload. These characteristics make FaaS a candidate paradigm for the computing continuum [4]. Platform providers can run as many functions as possible, saving resources and maximizing profit. Furthermore, thanks to the lightweight functions, they can more easily schedule workload on heterogeneous nodes, even resource-constrained ones. Jointly, customers achieve better performance and availability [34]. Therefore, the "divide and conquer" approach relieves stakeholders from managing a monolithic infrastructure by guaranteeing fine-grained access to resources [40]. These characteristics facilitate the disaggregation of work units and allow a focus shift toward how to define Service Level Objectives (SLOs). Still, effectively balancing and coordinating system and infrastructure objectives through serverless is a hard problem [28]. Indeed, there is the need to increase the optimization of such platforms, from operationalization to intelligent placing, scaling, and routing [46]. Open-source serverless solutions, such as OpenFaaS or Knative, offer such flexibility. Still, they require appropriate scaling methods to provide resource-efficient and highly available deployment. Achieving these goals takes work and should be handled by the platform. As discussed earlier, developers and infrastructure engineers should only express their SLO intents (e.g., efficiency, availability), letting the platform autonomously adapt. These strategies are complex as they must operate in a dynamic environment where the infrastructure nodes change. Furthermore, they need to communicate to solve conflicting objectives. That brings us to find ways to manage that, which translates into the need for algorithms capable of handling this complexity in (near-) real-time.

In this direction, one of the main tasks to solve is to encode mechanisms that can transform applications and management objectives into actions understandable by methods that run on the infrastructure nodes, thus coordinating appropriate responses. Intent-based systems are gaining popularity in this regard. The aim is to translate high-level inputs from stakeholders, e.g., using natural language, into actions that some automated function can apply to the system to satisfy the objectives. This approach offers high-level system-wide objectives or goals that tell the system what to do rather than how to achieve it [38]. Despite the increasing interest for intent-based approaches for the computing continuum and first attempts towards it [29], [27], [51], most of the applications are limited to the area of networking world [49], [60]. The aspects that emerge from this overview are naturally weaving with machine learning advances, where novel mechanisms like few-shot learners or self-supervised systems can aid

2642-6587/23/$31.00 ©2023 IEEE
DOI 10.1109/SOSE58276.2023.00035

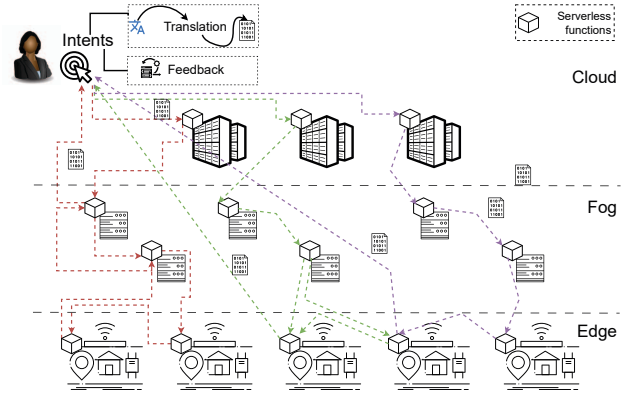


Fig. 1: Representation of the proposed framework. The aim is to set the foundation for handling the computing continuum complexity. We envision the possibility of achieving that through intent-based management of serverless functions.

in understanding and predicting complex dynamics in data. In particular, the computing continuum is a scenario where we must achieve a global equilibrium, balancing objectives, infrastructure changes, and dynamic behaviors [14]. Achieving this system-wide balance requires building a set of specialist models, the agents, that can examine the different aspects of the system [41]. However, these agents can only coexist if they "speak" the same language [32]. That is why having systems that can translate high-level objectives into "words" understandable by the agents in the system is essential.

To address this gap, we propose an architectural framework for intent-based management of serverless systems in the computing continuum. Our work focuses on identifying and implementing the components for translating stakeholder requirements into actions. Figure 1 depicts our proposal. We aim at having serverless functions controlling the infrastructure in the computing continuum and the applications running on top of it. We envision the computing continuum built by various geographical and regions that we want to coordinate and manage harmonically. We achieve this control through the translation of high-level intents into actions that the agents running in the serverless function will implement on the platform, e.g., by scaling the replicas of an application or by restructuring some infrastructure components. This task requires identifying an appropriate infrastructure and developing the correct components and their interconnection. In particular, the emphasis is on managing the agents that implement the actions needed to satisfy the SLOs using serverless functions for the computing continuum scenario, made of various regions. To this end, we propose a platform based on a three-layer intent-based architecture. We present a multi-objective use case in which we test our Proof of Concept (PoC), where conflicting SLOs must coexist, and the system needs to maintain its balance. To this end, we leverage a polynomial regression model to predict the system's behavior. We deploy the platform in Amazon's AWS cloud, showing how the proposed tools can integrate. We perform detailed simulations, hinting at how our approach can guarantee the resolution of conflicts in a complex system.

The structure of this paper is as follows. We investigate the focus of previous research and related work in Section II. Section III clarifies fundamental concepts and technologies needed for implementing such a system. In addition, we present our approach to specifying and working with intents, together with its architectural design and deployment on the public infrastructure provided by AWS. In Section IV, we evaluate the proposed Proof of Concept (PoC) through an extensive simulated environment. Later, in Section V, we discuss our contribution, pointing out both the positive outcomes and the limitations. Finally, Section VI concludes the paper.

For the sake of reproducibility and transparency, and to foster new developments, we publicly release the code and the data of our experiments. [1]

II. Related Work

A. Management frameworks for the computing continuum

Rohan Kumar et al. [30] focus on developing a framework for the computing continuum. They offer the integration of algorithms for executing management functions for handling function execution timing. To do that, they rely on federated function execution. However, their focus is not on conflicting objectives and the possible definition of intents. Other frameworks aim at doing that, focusing on specific functionalities for individual use cases. For example, Balouek et al. [6] propose a streaming data workflow and fast response framework. They aim to improve real-time data analytics for fast and effective earthquake management. Still discussing specific use cases, Peltonen et al. [43] propose an agenda for collaborative real-time learning in the Edge-cloud continuum in vehicular computing. Similarly, Torres et al. [54] propose a methodology for real-time monitoring and predictions using Kafka-ML. Zanella et al. [59] showsa framework for cloud continuum runtime management, focusing on dynamic and performance-aware task allocation policy. They, however, do not focus on intents or FaaS. Pilot-Edge, by Luckow et al. [31], provides a FaaS interface for application-level tasks, focusing on evaluating task placement. Ferrer et al. [17] envision a Cognitive cloud continuum with swarms for organizing the execution over the continuum. However, they do not focus on high-level SLOs. Pereira et al. [44] instead focus on the computing continuum availability guarantee, proposing a hierarchical approach. We take a different perspective, centering the framework on the intents and their definition and translation.

B. Intent-based systems

a) Translation: One of the most explored aspects of intent-based systems is the capability of translating intents expressed in natural language into rules applicable to the system [39], [21]. The current development in terms of language

[1] https://github.com/nspring00/three-layer-intent-based-networking-architecture-poc

models opens new frontiers in this direction. However, given the particular scenario, there is a need for ontologies and accurate models. For example, some work leverage predefined intent categories [24] or define specific languages [23], [57]. Other approaches leverage pre-trained models [33] or build intent translation on top of NLP algorithms [47], [52], [16].

b) Intent-driven platforms and frameworks: Mays et al. [36] specify resources and QoS requirements as non-functional intents for microservices. They formulate a minimization problem based on latency, CPU resources, and network traffic. Barrachina-Mu noz [7] focuses on end-to-end latency intent in Kubernetes. Mercian et al. [37] infer network intents from existing policies. Metsch et al. [38] enforce SLOs using intents in cloud-native deployments via Kubernetes, using an optimization approach and an A* algorithm. Kretsis et al. [29] propose an intent-driven edge-cloud platform called SERRANO. Kokkinos et al. [26] introduce a UI to create intent-based application requirements and propose a feedback-driven loop for workload optimization. Dzeparoska et al.[15] implement a management system that abstracts complexities using intents as inputs, featuring a policy abstraction and an API layer. Blum et al.[11] propose an SOA intent-based API for orchestrating NGNs, emphasizing collaboration with 3rd party services. Rafiq et al.[45] provide a user-friendly graphical interface and simplified input for end-to-end service orchestration and modeling of VNF descriptors. Paganelli et al.[42] present a two-layer network service description model for service provisioning and orchestration in SDN. Aklamanu et al.[2] automate network slicing through intents, while Abbas et al.[1] develop a network slicing platform with a GUI for inputting QoS requirements. Wang et al.[56] propose an intent-based "smart" slicing framework for vertical industries, and Chirivella-Perez et al.[13] present an E2E network slice management framework. Mascarenhas and Cruz [35] use an "intent-based" approach and autonomic computing concepts for autonomous cloud deployment management. Callegati et al.[12] propose a VIM POC for controlling SFC performance, and Beshley et al.[10] propose a switch migration approach considering QoS priorities.

C. Serverless solutions

Despite the advancement of serverless computing platforms, such as AWS Lambda [2], OpenFaaS [3], and other prototypes, only a few approaches currently allow stakeholders to define and implement objectives for the applications running on the platforms. For example, Klingler et al. [25] introduce code annotations that assist developers in optimizing function resource configuration, including memory allocation and addressing cold starts. At the same time, other approaches focus on reducing the resource footprint [9], [58] or managing resources at runtime, i.e., during function scaling and placement. In terms of location awareness, both commercial and open-source platforms offer limited support. Hu et al. [20] have developed a Domain-Specific Language (DSL) that allows developers to specify whether function execution should occur at the Edge or in the Cloud in a UAV system. Smith et al. [50] enable clients to indicate the execution area through invocation requests for network traffic optimization.

[2]https://aws.amazon.com/lambda

[3]https://www.openfaas.com

D. Gap and Opportunities

Whereas these results set good standards for managing distributed systems in a computing continuum, they are primarily limited to evaluating just a fragment of the overall scheme. Furthermore, most of the approaches focus on network problems. While it is true that internet networks embed much complexity and that VNFs have gained popularity, it is essential to expand this view on distributed systems at large. In fact, computing continuum systems face increased complexity due to the higher dynamic nature and the variety of applications and objectives connected to them. In addition, few proposals consider FaaS's necessity for opportune management; however, this characteristic is essential for us. Finally, most papers we report propose a PoC without deploying them in actual infrastructures. On the contrary, we test the solution on AWS. However, the scenario we build represents the majority of use cases, i.e., where the intent and business logic has to live in a public infrastructure with other application concurring for using resources.

III. Methodology

In our investigation, we focus on the design of a FaaS intent-based framework. First, it is essential to clarify that multiple technological and architectural approaches exist to designing intent-based systems, which mostly share a standard base structure. Their differences lie in the usage and interconnection of specific components and technologies. In our work, we choose the 3-layer architecture [60]. We further clarify why we chose this model and define its main components. Later, we present our implementation, clarifying the development choices. Lastly, in our case study, we set the main points for the evaluation we investigate in Section IV.

A. 3-layer Intent-based architectural approach

The 3-layer architecture, introduced by Zeydan [60], represents the most opportune model on top of which we can build our framework. The rationale is that it comprehends all the most relevant features, allowing flexibility in adapting the framework to various requirements. Indeed, due to its general features, this model already establishes a baseline for other architectures, such as the VIS [49] or end-to-end architecture [55]. That is why implementing and open-sourcing such an architecture represents an essential baseline for more complex applications for the computing continuum. In Figure 2, we depict the 3-layer architecture. We can see how we can develop a system that can interact with the stakeholders at the higher level. This top layer abstracts away the logic of managing a computing continuum system, where user requirements and observations collected from the infrastructure help the algorithms make decisions and perform actions on the infrastructure.

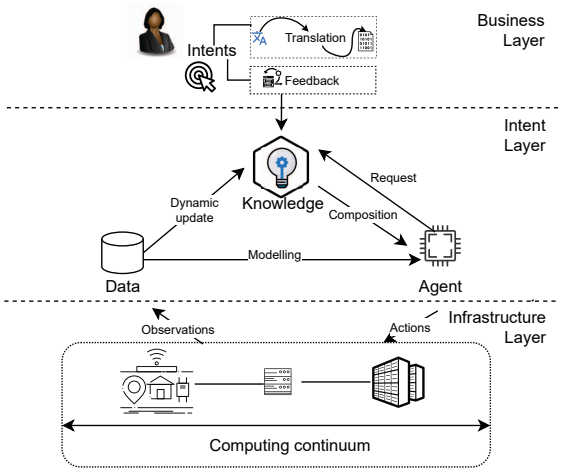


Fig. 2: Illustration of the 3-layer architecture with the main interaction of its components.

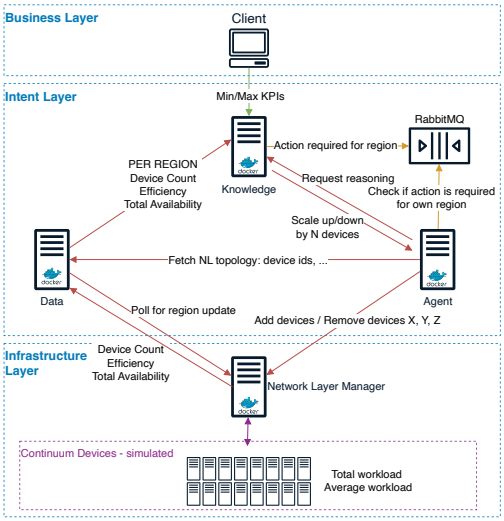


Fig. 3: Architecture overview of the PoC implementation.

a) Business Layer: Users can interact with the business layer to declare intents by specifying high-level metrics, i.e., an SLO, and triggering SLAs and other processes, goals, or objectives. Therefore, the business layer enables the high-level guidance of the whole system [60] and automatic detection and resolution of clashes between conflicting intents. As previously discussed, this layer can leverage language models for translation.

b) Intent Layer: The intent layer is essential, holding the logic for managing the computing continuum infrastructure. It primarily focuses on the re-evaluation and re-planning of the single steps during the execution of a sequence. These steps follow an accurate procedure. First, it detects what is happening in the system. Given this information and the user-defined intents, it defines some potential actions. Then, if it approves the action, it requests the procedure to apply it to the infrastructure. Multiple coupled components take care of managing this pipeline:

- the **Knowledge** component is responsible for processing and interpreting observations based on the defined intents. This component considers the relation between system elements, leveraging the actual state and performing inference.
- the **Agent** component is responsible for translating the intents received from the business layer into policies, acting as the communication interface towards the devices and transferring these translated policies to the infrastructure. In other words, it applies the decision outputted by the Knowledge.
- the **Data** component continuously observes the infrastructure nodes and has information on its topology and inventory. Additionally, they store the state of each intent, and upon any changes, the data model gets evaluated as accurately as possible. Therefore, upon any infrastructure state changes, they get forwarded to the Knowledge, and the updated state reaches the Agent.

c) Infrastructure Layer: The bottom layer is where physical nodes and computing continuum elements reside. The data about the infrastructure state gets abstracted into a formal representation before supplying it to the intent layer. It gets affected by the policies received from the Agent, which in this layer are further translated into concrete actions leveraging and translating abstract information on the hardware. The computing continuum system can be organized into multiple regions (or groups) depending on the infrastructure and or the services running on top of it. Therefore, we need to deploy as many Infrastructure layer managers as the number of regions. Albeit there might be regions, the computing continuum system must operate as a unique, organic entity,

B. Framework Implementation

The PoC implementation architecture (Figure 3) delineates the strategy of our interpretation of the 3-layer architecture. It consists of several components, each serving a specific purpose in the intent-based self-management system.

1) PoC implementation: First, it is worth reasserting that we do not focus on intent translation in this work. Therefore, the Business layer implementation focuses solely on defining the intents. From this point, the Knowledge component takes over and provides a RESTful API for intent management. Contextually, the Data component periodically fetches information about the infrastructure state. In this case, we use gRPC to request updates from the infrastructure nodes. The choice of gRPC allows excellent flexibility and performance, which is needed when dealing with a heterogeneous infrastructure. Once the Knowledge component processes the data, it sends a notification to the Agent, containing information about the eventual intent violation and triggering appropriate responses from the Agents. At the lower layer, an Infrastructure compo-

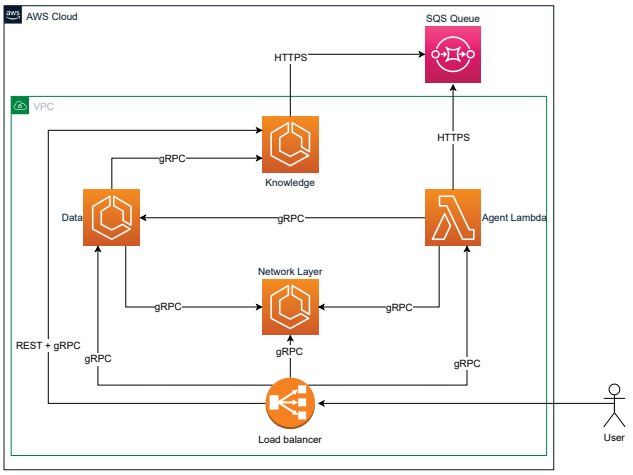


Fig. 4: Architecture of the Amazon Web Services (AWS) cloud deployment.

nent manages physical nodes and exposes gRPC services for data retrieval and command forwarding.

2) Deploying to the cloud: We implement the PoC to the AWS cloud due to its rich capabilities and tools. We implement the PoC using *C# 11*. [4]. The individual services rely on the *ASP.NET Core 6* framework [5] and on Docker containers, [6] The latter facilitates the deployment to the cloud and the services' orchestration during testing through *Docker compose*. Internally, the services communicate via Remote Procedure Calls (RPC) using gRPC. Figure 4 depicts the deployment. ECS takes care of the container orchestration. The messaging takes place through SQS. [7] The Agent component leverages *AWS Lambda*. Furthermore, Lambda function replicas are automatically scaled to the number of events that trigger the code execution, i.e., messages from Amazon Simple Queue Service (SQS) created by the Knowledge component. The user interface to the intent management leverages a RESTful API. All services are deployed inside an Amazon Elastic Container Service (ECS) cluster, which hosts a CloudMap instance responsible for internal and external service discovery using the defined service discovery entries. To conclude, the overall functioning translates to how we envision our PoC.

C. Evaluation

We simulate the dynamic behavior of the system's managed devices, loading resource usage data into the execution environment. We provide the simulation data to allow reproducibility. The specific dataset consists of a description of how the individually monitored metrics behave over time. Using a seeded random generator, we add a maximum of 5% jitter to each device simulation to ensure a more realistic environment.

[4]https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/

[5]https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-6.0

[6]https://docs.docker.com/get-started/overview/

[7]https://aws.amazon.com/sqs/

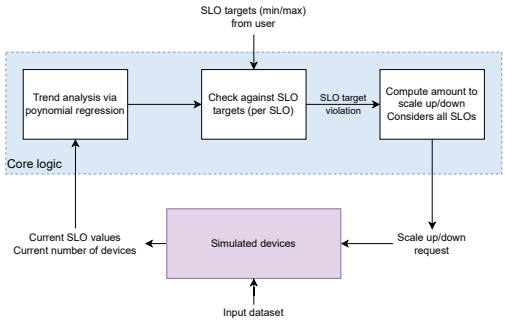


Fig. 5: Dataflow and core logic of the Knowledge component for the proposed case study simulation.

Since efficiency depends on the workload per device, and the device counts value at a specific time is unknown before executing the simulation, we specify CPU and memory targets as total workload values, which are then equally distributed across all devices. We express availability in the dataset using the average value. For simplicity, we assume that CPU and memory have the same values. Additionally, the currently active row in the dataset is iterated on each access. Indeed, every access to the dataset returns the next row. We conduct the simulation locally, using Docker Compose. Throughout the simulation, we consider just one region at a time, assuming regions are isolated and independent. In addition, one region can theoretically have several Network Layer managers, but we reduce this number to one during the simulation. This decision limits the ability to evaluate the Agent's decision process on which Network Layer manager to scale if there are multiple for one region. However, we leave this aspect to future analyses. Finally, to better frame the work, we keep the definition of intents static throughout the simulation. Future experiments can consider varying intents over time. In the following, we report the evaluation of the simulations where we consider efficiency and availability in isolation. Later, we combine the two KPIs and analyze the effect of conflicting intents.

IV. Case Study

In this case study, we aim to investigate how well the architecture works in real-life situations. The rationale is checking if different components can communicate efficiently and handle their respective tasks properly. We also examine how intents are specified and enforced and the deployment strategy. In the case study, we perform various simulations. First, we envision stakeholders defining conflicting intents (or SLOs), i.e., availability and efficiency, for their computing infrastructure. In the simulation, we focus on these intents, one region at a time, although our system design would support multiple regions. Figure 5 gives an overview of the experimental flow, which we detail in the results. The process includes creating and using simulated data and how the Knowledge component manages the output of the Agents' activity in the simulated architecture.

A. Intents definition

In general, intent-based systems support the declaration of predefined intents. Additionally, they are, for example, based on SLOs, which need to be implemented individually in the system. We limit the scope of the project, supporting two SLOs, *efficiency* and *availability*.

Efficiency is defined by the application's resource capacity, e.g., if it uses 50% of the available CPU capacity, CPU efficiency equates to 50%. We use a weighted sum of two different efficiencies to calculate the total system efficiency, which is used: CPU and memory. Although memory efficiency is relevant, computationally-bound tasks are more common. Therefore, in our paper, the CPU efficiency makes up 70% of the total efficiency weight. Further analyses, albeit essential, are out of the scope of this paper. Thus, $EFF = 0.3 * EFF_{MEM} + 0.7 * EFF_{CPU}$ gives the total system efficiency. Although EFF_{min} and EFF_{max} can be at most 1, the calculated EFF might exceed 100% because the workload can be more than the current number of devices can process, as the PoC implementation assumes that a device has precisely the capacity to manage one workload.

Availability is the second crucial metric. We measure it by dividing the time the system operates by the overall time. For example, during 10 hours of operation, the system shuts down for 30 minutes, achieving $\frac{9.5}{10} = 95\%$ availability. The system can replicate the service instances to increase its total availability. The availability increases with the number of service replicas r and can be calculated using the average availability [19] $AV = 1 - (1 - AV_{avg})^r$. For example, if a single service has 95% availability, three replicas would increase the service's availability to around 99.988%. As described before, the number of SLOs is limited by the scope of the implementation. However, using two separate SLOs still allows for studying conflict resolutions.

B. Inference model for the Knowledge component

Our strategy to implement these intents first analyzes the system's current state to predict how the monitored values will develop. This strategy compares the predicted results against the intents, deciding the number of active devices which fulfill the minimum and maximum intended target intent values. We predict the next step value for the SLOs by modeling the available data as a second-degree polynomial via polynomial regression, taking the last five average values of the monitored resources per region. The decision to consider the last five time stamps get considered stems from an empirical evaluation in the early phase of setting up the evaluation. We choose second-degree polynomial regression over the linear and the third-degree polynomial approaches as it appropriately models the oscillation of workload and availability during the simulation execution [3], [22]. Predicting the future resource values serves to extract the number of devices the system can support. Each SLO has its strategy for computing the Device Count (DC). In particular, for **Efficiency**: Let $0 <= EFF_{min}, EFF_{max} <= 1$ denote the boundary values specified by the intents, while DC_{cur} denotes the current DC. The DC bounds are given by:

$$DC_{min} = \lceil \frac{DC_{cur} * EFF_{cur}}{EFF_{max}} \rceil$$

$$DC_{max} = \lfloor \frac{DC_{cur} * EFF_{cur}}{EFF_{min}} \rfloor$$

For example, assume $DC_{cur} = 300$ and $EFF_{cur} = 70\%$. A minimum target efficiency of $EFF_{min} = 80\%$ would set the upper bound $DC_{max} = 262$. Note, that DC_{min} actually depends on EFF_{max}, while DC_{max} is influenced by DC_{max}.

For **availability**: Let $0 <= AV_{min}, AV_{max} <= 1$ denote the boundary values specified by the intents, and AV_{avg} the average availability trend. Additionally, an extremely small $\epsilon > 0$ is introduced to avoid computing $\log(0)$. The DC bounds are given by:

$$DC_{min} = \lceil \frac{\log(1 + \epsilon - AV_{min})}{\log(1 + \epsilon - AV_{avg})} \rceil$$

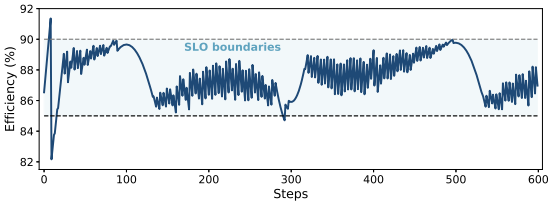
$$DC_{max} = \lfloor \frac{\log(1 + \epsilon - AV_{max})}{\log(1 + \epsilon - AV_{avg})} \rfloor$$

Again, an example can help understand the bigger picture. Unlike efficiency, the availability metric is not dependent on the DC because it uses the average availability. Assume $AVcur = 55\%$ and $AV_{min} = 99.9\%$. The goal requires at least $DC_{min} = 9$ replicas.

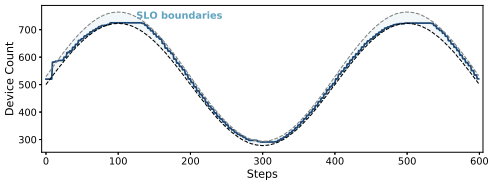
If all bounds are compatible, i.e., there is at least one value in the range of all bounds, the system takes its decision. If there are conflicting bounds, this conflict must be resolved while creating the slightest deviation possible from the intended state. Our approach is to prioritize minimum DC targets over maximum bounds. Due to this fact, the largest minimum DC bound is chosen as the new DC target. We express this target as $DC_{target} = \max(DC_{EFF_{min}}, DC_{AV_{min}})$, where $DC_{EFF_{min}}$ and $DC_{AV_{min}}$ denote the minimum device bounds of efficiency and availability respectively. In this way, our approach prioritizes the maximum efficiency bound. This strategy works well because the maximum efficiency threshold should not be exceeded, preventing extreme device loads. At the same time, a lower value, like in the availability scenario, only leads to more costs.

C. Simulation

1) Efficiency simulation: Efficiency is the first SLO we evaluate. In this simulation, we determine the total workload at each step and use it to calculate the final efficiency per device. The minimum efficiency target controls the maximum tolerated number of devices, as increased devices would lower the efficiency. The maximum efficiency target works vice-versa. The workload behavior follows a sinus-shaped curve of the form $WL(t) = sin(\frac{2\pi}{400} * t) * 200 + 450$. That means one cycle has 400 steps. The Data component updates the Knowledge after every fourth update cycle, i.e., 100 total updates and up to 100 total changes in the number of devices per cycle. For this evaluation, the minimum and maximum efficiency intent

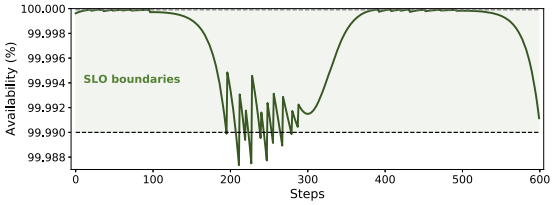


(a) Efficiency behavior.

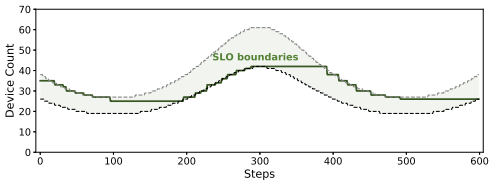


(b) Device Count behavior.

Fig. 6: Efficiency simulation



(a) Availability behavior.



(b) Device Count behavior.

Fig. 7: Availability simulation

values are $Eff_{min} = 85\%$ and $Eff_{max} = 90\%$. The rationale is that they let the system operate efficiently while keeping extra resources left.

Figure 6a depicts the minimum and maximum efficiency values as grey horizontal lines and the light blue area. The efficiency values over time are in blue. After an initial adaptation phase, the system keeps the efficiency between the given bounds. We can notice the points where the Knowledge scales the device pool, i.e., when efficiency gets close to the lower bound. Furthermore, the system decides not to scale if this is unnecessary. The following scaling only occurs when the efficiency gets close to the bound opposite to the one from before.

In Figure 6b, we can examine the variation of the device count number. The progression is bounded by the minimum and maximum Device Count (DC) in dark and light grey, and the light blue area. These last two curves directly correlate with the development of the workload WL, as $DC_{min} = \frac{WL}{Eff_{max}}$ and $DC_{max} = \frac{WL}{Eff_{min}}$. This plot better shows the efficiency development over time and the actual DC during execution time. The scaling operations of the Knowledge are visible, as the device count only changes when it hits DC_{max}. In conclusion, the system effectively enforces efficiency intents on its managed devices. Outside of the startup phase, the efficiency is always within bounds.

2) Availability simulation: The second simulation inspects the availability SLO. Availability AV measures the uptime of a system given requests coming from the applications. In this case, the maximum availability target affects the maximum tolerated DC, so accordingly for the minimum target.

The availability simulation follows a sinus-shaped curve of the form $AV(t) = sin(\frac{2\pi}{400} * t) * 0.1 + 0.3$, similarly to the previous experiment. The average availability of the system alternates between 20% and 40%. Although these numbers appear low, they still resemble a valid scenario where mobile devices can lose connectivity. This simulation value better highlights the effects on availability when increasing the DC. For example, a system with an average availability of 40% and five replicas already has around 92.2% total availability. In contrast, an average availability of 95% would lead to a value of around 99.99997% with the same number of devices. This evaluation's target availability is between $AV_{min} = 99.99\%$ and $AV_{max} = 99.9999\%$. The idea is to target the high availability behavior of a system that must operate (almost) at all times. Even though we cannot reproduce the exact behavior, we obtain realistic results due to the 4-byte floating point precision [18].

Figure 7a shows the current total availability, marked in green, and its bounds in grey, with a light green area. The jitter generates minor random deviations since scaling and total availability depend on the trend. Nevertheless, the approach effectively limits the number of bounds violations and deviations from the intended behavior. The time between timestep 100 and 499 equals one complete cycle of the dataset. During this time, the total availability only exceeds the intended maximum 19 times and undercuts the minimum 23 times. Thus, out of 400 timesteps, only 42 slightly deviate, at most by an absolute value of about $2.65\text{e}{-5}$. Lastly, Figure 7b shows the DC development and its bounds during the availability evaluation. Similarly to the efficiency case, scaling is only performed when the DC exceeds or undercuts the respective bounds; otherwise, no action is performed. To conclude, simulating only availability constraints works very well in isolation. The targeted availability range can only sometimes be adhered to mainly due to the jitter and the low DC. However, the system quickly recovers and corrects this deviation.

3) Combining multiple SLOs: Combining multiple SLOs, efficiency Eff, and availability AV can show how the system behaves in a more complex scenario, where conflicts may appear. Due to the inherent complexity of the setup, we carry two simulations. First, we aim to understand the basics of using multiple intents, studying the case of synchronized behavior

for efficiency and availability. Later, we create the environment for conflicts to further aid in evaluating the resolution of conflicting intents. During the following two simulations, the intent targets are the following: the average efficiency should be between $Eff_{min} = 75\%$ and $Eff_{max} = 95\%$. The total availability should be between $AV_{min} = 99.99\%$ and $AV_{max} = 99.9999\%$.

a) Synchronous SLO behavior: For the "sync" simulation, the workload and average availability simulation are precisely the same as in the isolated scenarios. One of them needs to behave in a mirrored way to behave synchronously. We can achieve his behavior by offsetting the sinus cycle by half a cycle.

First, we analyze the actual DC and its bounds for efficiency and availability (see Figure 8a). The actual DC tries to stay within the minimum DC bound, which is our target behavior. Therefore, minimum DC bounds take priority over maximum DC bounds, with the largest minimum DC bound as the target. The system aims to scale operations only when necessary. In the timestep interval 170–440, as the workload increases, the system focuses on reaching the largest minimum DC ($DC_{min_{Eff}}$). When both metrics decrease, the DC remains stable until action is required to meet $DC_{max_{AV}}$ or $DC_{max_{Eff}}$. There is no conflict within this range, as the largest minimum DC bound is lower than the smallest maximum DC bound. A slight conflict arises between timesteps 50–150 and 450–550 due to the simulation restarting after timestep 400. In this range, $DC_{max_{Eff}}$ is lower than $DC_{min_{AV}}$, necessitating a higher number of devices to meet the minimum availability bounds. Looking at efficiency, it generally stays within the required range. However, around timestep 50 to 150, a conflict between the two SLOs leads to lower efficiency. The maximum efficiency DC bound is disregarded due to the higher importance of minimum availability DC, resulting in a temporary drop in efficiency until the system resolves the conflict. While the system prefers availability over Efficiency (Figure 8b), it does not degrade the performance too much. Furthermore, as shown in Figure 8c, availability remains within the intended bounds. Only occasionally, particularly after conflicts begin (e.g., around timestamp 65), there is a slight decrease in availability, amounting to approximately $3.48\text{e}{-5}$. Overall, the system performs as expected, keeping all SLOs within their bounds when no conflicts exist. If conflicts arise, it handles them according to the specified requirements.

b) Asynchronous SLO behavior: This scenario aims to simulate an asynchronous behavior of the two SLOs efficiency and availability compared to each other. Further, this simulation purposely generates conflicts between the two intents to analyze the system's behavior under this stress. Regarding the SLO behavior, the behavior of the average availability stays exactly the same like in the last simulation at $AV(t) = sin(\frac{2\pi}{400} * t) * 0.1 + 0.3$. On the other hand, the frequency of the workload behavior is doubled and the original offset gets removed. This is given by $WL(t) = sin(2 * \frac{2\pi}{400} * t) * 20 + 30$, where the amplitude and the span of results still stay the same as before.

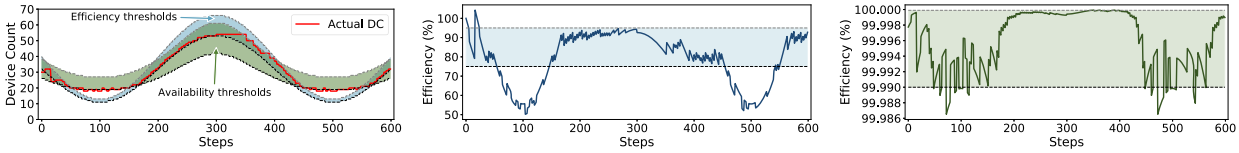
When looking at the development of the DC over time, visualized in Figure 9a, one can notice a suboptimal start. There is also an immediate conflict between efficiency and availability. Whereas the availability took precedence in the previous scenario, this time $DC_{min_{Eff}} > DC_{max_{AV}}$ around timestep 20 to 110. The system behaves as intended and scales the system to account for $DC_{min_{Eff}}$. Additionally, when the workload drops starting around timestep 50, the system underprovisions resources regarding efficiency because it detects a further drop in workload for the next timesteps. After a short period without conflicts around timesteps 110 to 125, $DC_{min_{AV}} > DC_{max_{Eff}}$ holds, and availability takes precedence over efficiency. The next conflict starts around timestep 220 and only takes a comparatively short amount of time, during which $DC_{min_{Eff}} > DC_{max_{AV}}$. Additionally, when the system resolves the conflict, one can notice the system's intended behavior to scale as late as possible. Figure 9b shows as well some of these conflicts. There are two significant drops in efficiency starting at timesteps 125 and 290, respectively. Both account for the timespans discussed above, where availability takes precedence over efficiency, and the Efficiency DC target is ignored. Nevertheless, efficiency almost always stays in its DC bounds. There is a light case of underprovisioning on the DC when the workload is sinking, and there is a conflict, which causes the efficiency to be too high for a short time, starting around timestamp 60 (or 460). Finally, Figure 9c shows the overall availability. At first sight, there is a drop in availability around timestep 360 beneath the minimum availability bound. Such deviations may happen primarily due to jitter. Still, the absolute value of such deviations is shallow. On the other hand, there are some increases in DC which exceed $DC_{max_{AV}}$ dramatically (see Figure 9a). This slight behavior happens when efficiency takes precedence over availability. Overall, the system fulfills its intended behavior. Isolated intents are fulfilled with minor temporary deviations, while more complex setups using multiple intents are also handled well, showing promising results.

V. Challenges and limitations

In this paper, we attempt to deploy a realistic PoC and offer a comprehensive case study. Still, many challenges are related to the prototype's design, implementation, or evaluation.

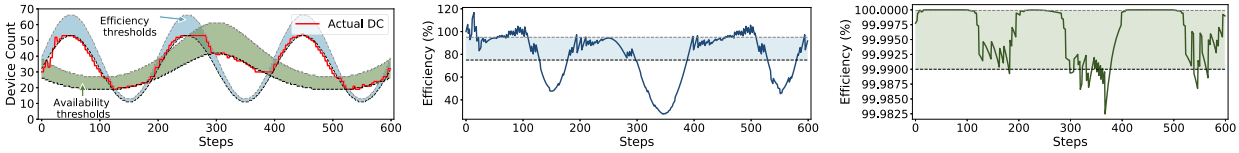
A. Case study setup

The framing of this work surrounds the implementation of the three-layer intent-based pattern. Therefore, we had to limit the scope of the paper to better focus on an in-depth analysis of the results. First, we do not deploy the system on an actual infrastructure. We perform the analysis on a simulated environment with a controlled amount of devices, which means no unexpected malfunctions, unanticipated anomalies, or energy consumption, instead essential in a real scenario [55]. The current setup is also homogeneous, i.e., every pod has the same CPU and memory capacity. Therefore, availability or efficiency measures are independent of the type of work



(a) Device Count behavior. (b) Efficiency behavior. (c) Availability behavior.

Fig. 8: Multi-SLO simulation (sync)



(a) Device Count behavior. (b) Efficiency behavior. (c) Availability behavior.

Fig. 9: Multi-SLO simulation (async)

currently done on the device, and there is no way of specifying refined requirements. Conversely, in a real scenario, with heterogeneous applications, there could be different device groupings, e.g., targeting specific classes of applications.

B. Intent specification and application

Another challenge is to deal with multiple conflicting intents, eliminating or weakening some of them while preserving the original meaning and achieving the best overall result. The number of negotiations increases when using a layered architecture. Therefore, the layers must also resolve conflicts between each other [60]. The given PoC simplifies these complex processes and executes all its reasoning and intent applications in the Knowledge component. Thus, there can not be any conflicts between layers, and no negotiations are necessary.

The system's support for the efficiency and availability SLOs provides an incomplete picture, requiring additional intent evaluation for behavior, application, and conflict resolution. However, intent-based systems are complex and tailored to specific use cases, limiting their reuse for other scenarios. Additionally, the number of supported intents is limited, as the system can only support intents with developer foresight and a limited learning aspect [48]. The efficiency intent is composed of fixed CPU and memory values, but future proposals could allow for dynamic metric selection with higher importance. In the simulation, availability does not depend on device utilization. We should also consider treating outliers differently. For example, we can remove them from the short record of the last measurements.

C. Framework implementation

Regarding implementing the framework itself, we had to limit the scope. Therefore, we do not consider sophisticated error handling, which would be essential in production. In addition, we primarily use synchronous gRPC for communication. Despite the excellent performance, we could bring loser coupling among components using asynchronous communication. We did not introduce resilience where we use messaging, i.e., SQS; additional measures, like Dead-Letter queues, could help filter out messages. Regarding orchestration, we could geographically distribute Agents to allow for less latency when communicating with the lower Layer managers, e.g., by assigning an Agent to a specific region. Currently, we use replication for adjusting the DC. However, we do not handle the scaling components replication, which is essential to prevent failures in production. Lastly, we use Docker Compose, but future versions could leverage Kubernetes.

D. Simulation

First and foremost, being a simulation, we do not react to actual application needs and infrastructure. The workload and availability values stored in the dataset are isolated from each other and can change arbitrarily. A richer model could address the two metrics being not independent and help inspect better algorithms for reasoning. We assume an evenly distributed workload across all devices, but this may only sometimes be true. If not, the system behaviors would require further adaptations. Furthermore, we limit the simulation to one region with a single Infrastructure Layer manager. While this does not directly impact the evaluation, it would be more valuable to see how the system handles multiple regions concurrently, e.g., checking how to distribute workload across multiple managers.

VI. Conclusions

Using intents can represent an essential step for controlling complex distributed systems, such as the computing continuum. This approach becomes even more crucial when using the serverless paradigm, which intrinsically holds intricacies in its management. In systems where the intents are first-class citizens, stakeholders can define what to do at a high level

without worrying about the details of how the infrastructure tackles that. In turn, the platform owners have more flexibility in managing the infrastructure due to the flexibility guaranteed by FaaS. This paper presented a detailed approach to designing and implementing an intent-based system for systems management, primarily based on the 3-layer architecture initially described in [60]. This PoC aims to create an intent-based solution for the computing continuum's main management challenges. Overall, the proposed framework achieves its goal of managing the infrastructure, i.e., the devices, effectively and reliably based on the high-level intents – efficiency and availability. The system reacts autonomously and executes self-correcting actions to satisfy the previously defined SLOs. In particular, we address intents specification and application in the PoC through a simplified device model. Furthermore, we design the whole application as a set of containerized microservices. Therefore, the PoC can be seamlessly deployed to the cloud and efficiently use managed services on cloud platforms like AWS. Future improvements to our work include adding complexity in the evaluation, testing our system in production environments, and making the deployment more resilient to work in production. Finally, exploring NLP algorithms for user intents translation would be valuable.

VII. Acknowledgments

We would like to sincerely thank Nikolaus Spring for his valuable contribution, and commitment to this paper. His dedicated work and experimentation were crucial in advancing our research and achieving meaningful results.

References

[1] Khizar Abbas, Muhammad Afaq, Talha Ahmed Khan, Adeel Rafiq, and Wang Cheol Song. Slicing the core network and radio access network domains through intent-based networking for 5G networks. *Electronics (Switzerland)*, 9(10):1–25, 2020.

[2] Fred Aklamanu, Sabine Randriamasy, Eric Renault, Imran Latif, and Abdelkrim Hebbar. Intent-based real-time 5g cloud service provisioning. In *2018 IEEE Globecom Workshops (GC Wkshps)*, pages 1–6. IEEE, 2018.

[3] Animesh Agarwal. Polynomial Regression. https://towardsdatascience.com/polynomial-regression-bbe8b9d97491, 2018. Accessed: 2022-10-26.

[4] Mohammad S Aslanpour, Adel N Toosi, Claudio Cicconetti, Bahman Javadi, Peter Sbarski, Davide Taibi, Marcos Assuncao, Sukhpal Singh Gill, Raj Gaire, and Schahram Dustdar. Serverless edge computing: vision and challenges. In *2021 Australasian Computer Science Week Multiconference*, pages 1–10, 2021.

[5] Ioana Baldini, Paul Castro, Kerry Chang, Perry Cheng, Stephen Fink, Vatche Ishakian, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, Aleksander Slominski, et al. Serverless computing: Current trends and open problems. *Research advances in cloud computing*, pages 1–20, 2017.

[6] Daniel Balouek-Thomert, Ivan Rodero, and Manish Parashar. Harnessing the computing continuum for urgent science. *ACM SIGMETRICS Performance Evaluation Review*, 48(2):41–46, 2020.

[7] Sergio Barrachina-Muñoz, Jorge Baranda, Miquel Payaró, and Josep Mangues-Bafalluy. Intent-based orchestration for application relocation in a 5g cloud-native platform. In *2022 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 94–95. IEEE, 2022.

[8] Ali Basiri, Niosha Behnam, Ruud De Rooij, Lorin Hochstein, Luke Kosewski, Justin Reynolds, and Casey Rosenthal. Chaos engineering. *IEEE Software*, 33(3):35–41, 2016.

[9] David Bermbach, Jonathan Bader, Jonathan Hasenburg, Tobias Pfandzelter, and Lauritz Thamsen. Auctionwhisk: Using an auction-inspired approach for function placement in serverless fog platforms. *Software: Practice and Experience*, 52(5):1143–1169, 2022.

[10] Mykola Beshley, Andrii Pryslupskyi, Oleksiy Panchenko, and Marian Seliuchenko. Dynamic switch migration method based on qoe-aware priority marking for intent-based networking. In *2020 IEEE 15th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET)*, pages 864–868. IEEE, 2020.

[11] Niklas Blum, Simon Dutkowski, and Thomas Magedanz. Insert an intent-based service request api for service exposure in next generation networks. In *2008 32nd Annual IEEE Software Engineering Workshop*, pages 21–30. IEEE, 2008.

[12] Franco Callegati, Walter Cerroni, Chiara Contoli, and Francesco Foresta. Performance of intent-based virtualized network infrastructure management. In *2017 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2017.

[13] Enrique Chirivella-Perez, Pablo Salva-Garcia, Ruben Ricart-Sanchez, Jose Alcaraz Calero, and Qi Wang. Intent-based E2E network slice management for industry 4.0. *2021 Joint European Conference on Networks and Communications and 6G Summit, EuCNC/6G Summit 2021*, pages 353–358, 2021.

[14] Schahram Dustdar, Victor Casamayor Pujol, and Praveen Kumar Donta. On distributed computing continuum systems. *IEEE Transactions on Knowledge and Data Engineering*, 35(4):4092–4105, 2022.

[15] Kristina Dzeparoska, Nasim Beigi-Mohammadi, Ali Tizghadam, and Alberto Leon-Garcia. Towards a self-driving management system for the automated realization of intents. *IEEE Access*, 9:159882–159907, 2021.

[16] Adriana Fernandez-Fernandez, Estefania Coronado, Alberto Erspamer, Georgios Samaras, Vasileios Theodorou, and Shuaib Siddiqui. Unlocking the path towards intelligent telecom marketplaces for beyond 5g and 6g networks. *IEEE Communications Magazine*, 2023.

[17] Ana Juan Ferrer, Sören Becker, Florian Schmidt, Lauritz Thamsen, and Odej Kao. Towards a cognitive compute continuum: an architecture for ad-hoc self-managed swarms. In *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pages 634–641. IEEE, 2021.

[18] David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM computing surveys (CSUR)*, 23(1):5–48, 1991.

[19] Michael Haken. Availability and beyond: Understanding and improving the resilience of distributed systems on aws. Technical report, Amazon Web Services, November 2021.

[20] Justin Hu, Ariana Bruno, Brian Ritchken, Brendon Jackson, Mateo Espinosa, Aditya Shah, and Christina Delimitrou. Hivemind: A scalable and serverless coordination control platform for uav swarms. *arXiv preprint arXiv:2002.01419*, 2020.

[21] Jiaorui Huang, Chungang Yang, Shiwen Kou, and Yanbo Song. A brief survey and implementation on ai for intent-driven network. In *2022 27th Asia Pacific Conference on Communications (APCC)*, pages 413–418. IEEE, 2022.

[22] IBM Corporation. What is linear regression? https://www.ibm.com/topics/linear-regression#:~:text=Resources-,What%20is%20linear%20regression%3F,is%20called%20the%20independent%20variable., 2022. Accessed: 2022-09-14.

[23] Arthur Selle Jacobs, Ricardo José Pfitscher, Ronaldo Alves Ferreira, and Lisandro Zambenedetti Granville. Refining network intents for self-driving networks. In *Proceedings of the Afternoon Workshop on Self-Driving Networks*, pages 15–21, 2018.

[24] Mariam Kiran, Eric Pouyoul, Anu Mercian, Brian Tierney, Chin Guok, and Inder Monga. Enabling intent to configure scientific networks for high performance demands. *Future Generation Computer Systems*, 79:205–214, 2018.

[25] Raffael Klingler, Nemanja Trifunovic, and Josef Spillner. Beyond@ cloudfunction: Powerful code annotations to capture serverless runtime patterns. In *Proceedings of the Seventh International Workshop on Serverless Computing (WoSC7) 2021*, pages 23–28, 2021.

[26] Panagiotis Kokkinos, Dionisis Margaris, and Dimitris Spiliotopoulos. A quality of experience illustrator user interface for cloud provider recommendations. In *HCI International 2022 Posters: 24th International Conference on Human-Computer Interaction, HCII 2022, Virtual Event,*

June 26–July 1, 2022, Proceedings, Part I, pages 308–315. Springer, 2022.

[27] Henna Kokkonen, Lauri Lovén, Naser Hossein Motlagh, Juha Partala, Alfonso González-Gil, Ester Sola, Iñigo Angulo, Madhusanka Liyanage, Teemu Leppänen, Tri Nguyen, et al. Autonomy and intelligence in the computing continuum: Challenges, enablers, and future directions for orchestration. *arXiv preprint arXiv:2205.01423*, 2022.

[28] Samuel Kounev, Cristina Abad, Ian Foster, Nikolas Herbst, Alexandru Iosup, Samer Al-Kiswany, Ahmed Ali-Eldin Hassan, Bartosz Balis, André Bauer, Andre Bondi, et al. Toward a definition for serverless computing. 2021.

[29] Aristotelis Kretsis, Panagiotis Kokkinos, Polyzois Soumplis, Juan Jose Vegas Olmos, Marcell Fehér, Márton Sipos, Daniel E Lucani, Dmitry Khabi, Dimosthenis Masouros, Kostas Siozios, et al. Serrano: transparent application deployment in a secure, accelerated and cognitive cloud continuum. In *2021 IEEE International Mediterranean Conference on Communications and Networking (MeditCom)*, pages 55–60. IEEE, 2021.

[30] Rohan Kumar, Matt Baughman, Ryan Chard, Zhuozhao Li, Yadu Babuji, Ian Foster, and Kyle Chard. Coding the computing continuum: Fluid function execution in heterogeneous computing environments. In *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 66–75. IEEE, 2021.

[31] Andre Luckow, Kartik Rattan, and Shantenu Jha. Pilot-edge: Distributed resource management along the edge-to-cloud continuum. In *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 874–878. IEEE, 2021.

[32] Xuewen Luo, Hsiao-Hwa Chen, and Qing Guo. Semantic communications: Overview, open issues, and future research directions. *IEEE Wireless Communications*, 29(1):210–219, 2022.

[33] Hocine Mahtout, Mariam Kiran, Anu Mercian, and Bashir Mohammed. Using machine learning for intent-based provisioning in high-speed science networks. In *Proceedings of the 3rd International Workshop on Systems and Network Telemetry and Analytics*, pages 27–30, 2020.

[34] Anupama Mampage, Shanika Karunasekera, and Rajkumar Buyya. A holistic view on resource management in serverless computing environments: Taxonomy and future directions. *ACM Computing Surveys (CSUR)*, 54(11s):1–36, 2022.

[35] Manuel Duarte Mascarenhas and Rui Santos Cruz. Int2it: An intent-based tosca it infrastructure management platform. In *2022 17th Iberian Conference on Information Systems and Technologies (CISTI)*, pages 1–7. IEEE, 2022.

[36] AL-Naday Mays, Tom Goethals, and Bruno Volckaert. Intent-based decentralized orchestration for green energy-aware provisioning of fog-native workflows. In *2022 18th International Conference on Network and Service Management (CNSM)*, pages 184–190. IEEE, 2022.

[37] Anu Mercian, Faraz Ahmed, Puneet Sharma, Shaun Wackerly, and Charles Clark. Mind the semantic gap: Policy intent inference from network metadata. In *2021 IEEE 7th International Conference on Network Softwarization (NetSoft)*, pages 312–320. IEEE, 2021.

[38] Thijs Metsch, Magdalena Viktorsson, Adrian Hoban, Monica Vitali, Ravi Iyer, and Erik Elmroth. Intent-driven orchestration: Enforcing service level objectives for cloud native deployments. *SN Computer Science*, 4(3):268, 2023.

[39] Gilbert Moisio, Alexandre Gonzalvez, and Noam Zeitoun. Introduction to the artificial intelligence that can be applied to the network automation journey. *arXiv preprint arXiv:2204.00800*, 2022.

[40] Gabriele Morabito, Christian Sicari, Armando Ruggeri, Antonio Celesti, and Lorenzo Carnevale. Secure-by-design serverless workflows on the edge–cloud continuum through the osmotic computing paradigm. *Internet of Things*, 22:100737, 2023.

[41] Andrea Morichetta, Victor Casamayor Pujol, and Schahram Dustdar. A roadmap on learning and reasoning for distributed computing continuum ecosystems. In *2021 IEEE International Conference on Edge Computing (EDGE)*, pages 25–31. IEEE, 2021.

[42] Federica Paganelli, Francesca Paradiso, Monica Gherardelli, and Giulia Galletti. Network service description model for vnf orchestration leveraging intent-based sdn interfaces. In *2017 IEEE Conference on Network Softwarization (NetSoft)*, pages 1–5. IEEE, 2017.

[43] Ella Peltonen, Arun Sojan, and Tero Päivärinta. Towards real-time learning for edge-cloud continuum with vehicular computing. In *2021 IEEE 7th World Forum on Internet of Things (WF-IoT)*, pages 921–926. IEEE, 2021.

[44] Paulo Pereira, Carlos Melo, Jean Araujo, Jamilson Dantas, Vinícius Santos, and Paulo Maciel. Availability model for edge-fog-cloud continuum: an evaluation of an end-to-end infrastructure of intelligent traffic management service. *The Journal of Supercomputing*, pages 1–28, 2022.

[45] Adeel Rafiq, Asif Mehmood, Talha Ahmed Khan, Khizar Abbas, Muhammad Afaq, and Wang-Cheol Song. Intent-based end-to-end network service orchestration system for multi-platforms. *Sustainability*, 12(7):2782, 2020.

[46] Philipp Raith, Stefan Nastic, and Schahram Dustdar. Serverless edge computing—where we are and what lies ahead. *IEEE Internet Computing*, 27(3):50–64, 2023.

[47] Mohammad Riftadi, Jorik Oostenbrink, and Fernando Kuipers. Gp4p4: Enabling self-programming networks. *arXiv preprint arXiv:1910.00967*, 2019.

[48] Ilja Shmelkin, Daniel Matusek, Tim Kluge, Thomas Springer, and Alexander Schill. Intent-based adaptation coordination of highly decentralized networked self-adaptive systems. In Mikhailo Klymash, Mykola Beshley, and Andriy Luntovskyy, editors, *Future Intent-Based Networking*, pages 69–100, Cham, 2022. Springer International Publishing.

[49] Amritpal Singh, Gagangeet Singh Aujla, and Rasmeet Singh Bali. Intent-based network for data dissemination in software-defined vehicular edge computing. *IEEE Transactions on Intelligent Transportation Systems*, 22(8):5310–5318, 2021.

[50] Christopher Peter Smith, Anshul Jindal, Mohak Chadha, Michael Gerndt, and Shajulin Benedict. Fado: Faas functions and data orchestrator for multiple serverless edge-cloud clusters. In *2022 IEEE 6th International Conference on Fog and Edge Computing (ICFEC)*, pages 17–25. IEEE, 2022.

[51] Hui Song, Ahmet Soylu, and Dumitru Roman. Towards cognitive self-management of iot-edge-cloud continuum based on user intents. In *2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC)*, pages 1–4. IEEE, 2022.

[52] Sondes Bannour Souihi, Hai-Anh Tran, Sami Souihi, et al. When nlp meets sdn: an application to global internet exchange network. In *ICC 2022-IEEE International Conference on Communications*, pages 2972–2977. IEEE, 2022.

[53] Jörg Thalheim, Antonio Rodrigues, Istemi Ekin Akkus, Pramod Bhatotia, Ruichuan Chen, Bimal Viswanath, Lei Jiao, and Christof Fetzer. Sieve: Actionable insights from monitored metrics in distributed systems. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*, pages 14–27, 2017.

[54] Daniel R Torres, Cristian Martín, Bartolomé Rubio, and Manuel Díaz. An open source framework based on kafka-ml for distributed dnn inference over the cloud-to-things continuum. *Journal of Systems Architecture*, 118:102214, 2021.

[55] Luis Velasco, Marco Signorelli, Oscar González De Dios, Chrysa Papagianni, Roberto Bifulco, Juan Jose Vegas Olmos, Simon Pryor, Gino Carrozzo, Julius Schulz-Zander, Mehdi Bennis, Ricardo Martinez, Filippo Cugini, Claudio Salvadori, Vincent Lefebvre, Luca Valcarenghi, and Marc Ruiz. End-to-end intent-based networking. *IEEE Communications Magazine*, 59(10):106–112, 2021.

[56] Dong Wang, Ruiran Su, and Shenhu Zhang. An intent-based smart slicing framework for vertical industry in b5g networks. In *2021 IEEE/CIC International Conference on Communications in China (ICCC Workshops)*, pages 389–394. IEEE, 2021.

[57] Tomasz Winiarski, Wojciech Dudek, Maciej Stefańczyk, Łukasz Zieliński, Daniel Giełdowski, and Dawid Seredyński. An intent-based approach for creating assistive robots' control systems. *arXiv preprint arXiv:2005.12106*, 2020.

[58] Rich Wolski, Chandra Krintz, Fatih Bakir, Gareth George, and Wei-Tsung Lin. Cspot: Portable, multi-scale functions-as-a-service for iot. In *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, pages 236–249, 2019.

[59] Michele Zanella, Filippo Sciamanna, and William Fornaciari. Barman: A run-time management framework in the resource continuum. *Sustainable Computing: Informatics and Systems*, 35:100663, 2022.

[60] Engin Zeydan and Yekta Turk. Recent advances in intent-based networking: A survey. In *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*, pages 1–5, 2020.