Invited Paper: DeepSLOs for the Computing Continuum

Victor Casamayor Pujol
Boris Sedlak
Yanwei Xu
Distributed Systems Group, TU Wien,
Vienna 1040, Austria
{vcasamayor,bsedlak,y.xu}@dsg.tuwien.ac.at

Praveen Kumar Donta
Schahram Dustdar
Distributed Systems Group, TU Wien,
Vienna 1040, Austria
{pdonta,dustdar}@dsg.tuwien.ac.at

ABSTRACT

The advent of the computing continuum, i.e., the blending of all existing computational tiers, calls for novel techniques and methods that consider its complex dynamics. This work presents the DeepSLO as a novel design paradigm to define and structure Service Level Objectives (SLOs) for distributed computing continuum systems. Hence, when multiple stakeholders are involved, the DeepSLO allows them to plan the overarching behaviors of the system. Further, the techniques employed (Bayesian networks, Markov blanket, Active inference) provide autonomy and decentralization to each SLO while the DeepSLO hierarchy remains to account for objectives dependencies. Finally, DeepSLOs are represented graphically, as well as individual SLOs enabling a human interpretation of the system performance.

CCS CONCEPTS

• **Software and its engineering → Software design engineering**.

ACM Reference Format:

Victor Casamayor Pujol, Boris Sedlak, Yanwei Xu, Praveen Kumar Donta, and Schahram Dustdar. 2024. Invited Paper: DeepSLOs for the Computing Continuum. In *Advanced Tools, Programming Languages, and PLatforms for Implementing and Evaluating algorithms for Distributed systems (ApPLIED'24), June 17, 2024, Nantes, France.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3663338.3663681

1 INTRODUCTION

The Computing Continuum (CC) is a new computing paradigm that is steadily emerging and promising to link the huge computational capabilities of the Cloud with the proximity and low latency of the Edge. Cloud capabilities and its landscape of services and applications have driven how we interact and benefit from computing systems. For many years, the Cloud has been the only way to consume large-scale internet-based services [17]. However, society is exploring the benefits from new applications concerned with smart cities [11], autonomous driving [5], resource management [3], or e-health [2]. These applications set service requirements at a level the Cloud cannot fulfill.

This work is licensed under a Creative Commons Attribution International 4.0 License.

ApPLIED'24, June 17, 2024, Nantes, France
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0670-7/24/06
https://doi.org/10.1145/3663338.3663681

As an initial answer to that demand, Edge computing arose to decrease the latency of this next generation of internet-based services, reduce network bandwidth utilization by performing computations close to data sources, and by enhancing privacy by not storing users' data [29]. Regardless, more than Edge computing is needed to provide the computing, networking, and storage capabilities that the Cloud is giving. Hence, crucial to the CC is its ambition of blending all computing tiers, e.g., the Cloud, the Fog, the Edge, and the Internet of Things (IoT), to build Distributed Computing Continuum Systems (DCCS) [6], which leverage the best from each computing model.

Developing and managing DCCS brings new challenges that must be addressed. To start, the CC will be a multi-proprietary and multi-tenant computing environment, which implies that parts owned by different actors will be combined in a single system. One can imagine the system having at least one vendor for Cloud services, another for Edge services, the IoT layer covered by a third company, and the network provider also offering specific network-related functions to enhance critical services. Further, all these infrastructure components will be shared with other applications; thus, creating a multi-tenant environment, as it is currently occurring in the Cloud [1]. Hence, on the one side, there is a need to build appropriate and manageable solutions for all the system's stakeholders, meaning solutions that consider both the system's high-level needs and each of the stakeholders' lower-level specifications. On the other side, services might not always encounter their ideal environment because devices' capabilities and availability fluctuate over time; therefore, runtime adaptations need to be built into the system, such as changing data quality.

In addition, these systems are geographically distributed. Imagine an application being used in a region for smart city management; it will employ IoT devices where data is collected, constrained single-board computers (SBC) next to the IoT for light processing, edge servers in the larger cities for inference and low-latency servicing, and somewhere distant there will be a data center, providing Cloud functionalities. Indeed, if we foresee larger and more complex applications, the distribution is even more accentuated. This property challenges the deployment and adaptation capabilities of the services in the infrastructure, leading toward the need of decentralized techniques. Interestingly, decentralization is well aligned with the needs associated in a multi-proprietary and multi-tenant system, but it is opposite to the main trends for the Cloud current business. Fortunately, from the research community there are important voices calling for multi-proprietary Cloud environments [31].

In that regard, current Cloud-based providers use Service Level Objectives (SLOs) to offer application owners guarantees of performance through Service Level Agreements (SLAs) [13, 32, 37].

Briefly, SLOs are constraints to specific Services Level Indicators (SLI), which are measurable system metrics such as the CPU usage. The provider must fulfill its constraints as part of the SLA; however, if the SLO is violated by exceeding the SLO limits, the provider will have to compensate its client, e.g., by paying a penalty. Hence, deployment and adaptation operations are linked to the fulfillment of the selected SLO, as a bilateral agreement and considering only the performance on the provider's side. Clearly, in a multi-proprietary scenario, where infrastructure components are related but belong to different providers, the SLOs selected will need to consider the performance on several providers' ends and the possible ramifications of SLO violations. Hence, acknowledging propagation effects is fundamental to equip DCCS with accountability capacity, which is crucial to reach agreement among stakeholders.

In the forthcoming CC paradigm, the current approach to management through an SLO falls short. First, when considering applications consisting of a set of services, a single SLO is incapable to properly hinting the adequate adaptation operation to correct the application's performance. Workarounds exist, for example Qiu et al. [21] compute the critical path over the application's services to detect and act on the service that might violate the SLO. However, finding the critical path to detect such issues have a couple of drawbacks: (1) computing the critical path is neither a fast nor simple task, although there are some approaches into huge applications [40]. (2) identifying the service that is producing the delay is not enough to pinpoint its cause, possibly leading to ill-posed adaptations. Another common approach to ensure SLO fulfillment for Cloud-based applications with many services is finding the root-cause of the SLO violation. In that regard, a good example is the work of Chen et al. [4].

Unfortunately, the underlying problem is the same as before and it is not tackled: a limited and centralized visibility into the system's structure and performance. Further, most of solutions react only when the SLO has been violated, which can be too late for a recovery in large, distributed, and complex systems. Other works aim at using high-level SLOs with the purpose of gaining more accurate information on system's behavior and to achieve a proactive adaptation by means of deep learning techniques [16]. However, the definition of the high-level SLOs, their linking to lower-level system metrics, and how to consequently adapt the system are still open questions under research. Further, when discussing about adaptation the Kubernetes autoscaler is the state-of-the-art solution for Cloud-based deployments. However, when moving to the Edge or when the scenario is the CC, autoscaling is not sufficient due to the limited resources of the devices. Hence, solutions considering offloading or measures that can trade-off service performance with quality or cost are needed to be explored [27]. Hence, new solutions for DCCS require precise specification and visibility on the system to have efficient and effective adaptation capabilities.

The vanilla method to provide a sufficient level of visibility and to have a fully specified system is generating SLOs for each service. This way, whenever an SLO is violated, the affected service can be identified immediately and the measures for the adaptation can be tailored to the service. Indeed, this is seen as an overhead for any Cloud-based application as it requires collecting and centrally analyzing the system's data and to timely respond if required. It implies building an ad-hoc network to transfer all information without affecting the application behavior, which has to be fast enough to maximize system's availability. Regardless, we all have witnessed the success of Cloud applications, which work generally well with its current, and more simple, management approach.

However, the CC with its geographic distribution, decentralization, and multi-tenancy, leads to the specification of all services as the only approach to properly develop an accountable system for all its participants. Further, a complete services specification aids keeping the system functionality as expected. Nevertheless, specifying SLOs for each service requires a detailed study of their interconnections. The system's information shared between services has to be carefully measured (for performance, security, and privacy issues) sending only the essential data for a service to evaluate its SLO compliance and to apply the most appropriate elasticity strategy. Further, this means that each service requires a certain level of autonomy being able to take care of its SLO with the minimum interaction with other services or other higher-level entities (e.g., the orchestrator or other meta-services). Hence, each service and its SLO (or SLOs) becomes a single accountable entity. It is crucial to clarify that this does not entail service providers incorporating all of this logic into the service itself, but rather, the service is required to expose the necessary interfaces for the SLOs.

In this article, we propose the DeepSLO as a design paradigm. A DeepSLO encompasses concepts, structures, and a mathematical framework as the cornerstone to define and specify CC applications. The DeepSLO is an overarching structure for a CC application. It connects conditionally dependent SLOs by building it as a Bayesian network. The Bayesian network provides the proper mathematical framework to predict their behavior, to allow any further system optimization, and to study trade-offs between SLOs. Further, the Bayesian model is interpretable, which means that its outcomes are explainable. The DeepSLO provides a deep (hierarchical) structure between the requirements (SLOs), allowing their interconnection and conflict resolution. These aspects enable the integration of all stakeholders, by letting them specify service-device requirements and providing a hierarchical structure to agree and prioritize interactions between each SLO. The SLOs within the DeepSLO are used to fully describe the application's requirements, considering both the services they control and the devices where they are deployed. Each SLO has its area of interest, which means that all system variables that affect the SLO are considered, but all the others are removed from the analysis as a causal filter. Further, SLOs are given means to act autonomously with active inference, and hence, they can plan the best strategy to keep SLOs fulfilled. Developing the SLOs as intelligent agents with a narrow set of system variables to look at aids their decentralization and autonomous SLO-based management.

In the following sections, we will provide a bottom-up description of the DeepSLO. We take this perspective because some fundamental characteristics of the DeepSLO naturally emerge from its components. Further, to clarify the explanations given at each section a running example will follow them.

2 SLOS AS SERVICE REQUIREMENTS

Applications can be composed of a large variety of services. Indeed, depending on the application, services will vary. However, it is reasonable to think that services can be grouped or profiled, severely reducing their diversity according to their specific computational demands. Simply put, services requiring high CPU utilization would be a different category than those that are doing I/O operations, and services requiring a GPU would be another. The granularity of these categories has to be helpful to identify services while keeping the number of categories at a manageable level [15]. For the sake of clarity, we assume that services providing similar functionalities have similar computational demands. If such an assumption is refuted, it would be done based on their computational demands instead of classifying services by functionality, as previously explained.

Understanding SLOs as service requirements means that each service has to be associated with SLOs that guarantee its expected behavior. The complexity of the service can demand more than one SLO; hence, it is fundamental that the SLOs properly reflect the service's requirements. Having defined types of services can ease the identification of their required SLOs. Interestingly, this expected behavior of the service implies some knowledge of the type of device that will be hosting it. If we look into current Cloud-based SLOs, they do not need to consider the device in which it will be deployed to adjust the SLO. Actually, if the service requires a specific piece of hardware, e.g., a GPU, this constraint is sent to the scheduler. However, in this new CC paradigm, having a single scheduler (or resource manager) managing all hardware resources is impossible due to their geographical distance or the different ownership of the computing resources. Hence, adding this type of constraint to the service, e.g., a specific hardware need, gives the flexibility that, regardless of the final geographical location of the service in the continuum, the local resource manager will be aware of the service requirements.

In addition, the continuum consists of a large variety of heterogeneous devices. Hence, expecting the same SLO compliance rate for the same service deployed in the Cloud as in a constraint edge device is unrealistic. In that regard, SLOs need to be aware of the possible deployment options for the service. Similarly, hosting devices can be grouped or classified with respect to their characteristics [20], as has been done with the services. Hence, in situations where the behavior of the service is known for a specific device type, it is possible to infer its expected behavior in other types. Figure 1 shows the different components required to build SLOs as service requirements for the CC. In the following subsections, each of the aspects shown in the figure will be explained.

Running example. To start, consider a simple machine learning pipeline with three services: data gathering and pre-processing, model training, and inference. Further, imagine that these services are part of an eHealth application able to predict if the monitored person will suffer from an adverse medical condition and, if so, trigger the needed alarms to provide medical assistance as fast as possible. Defining the requirements of the services needs proper dedication. Still, the classification of services, e.g., data gathering or inference, already identifies which requirements can be meaningful for the service.

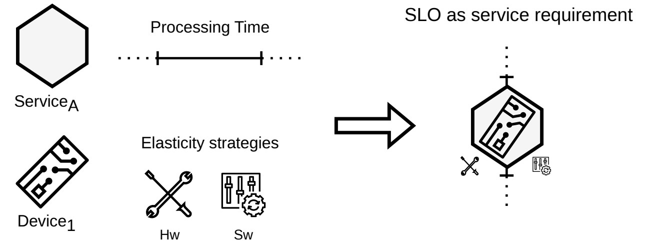

Figure 1: The SLO as a service requirement can be cast as a requirement (processing time) over service (A) hosted in device (1), which has elasticity strategies (Hw & Sw).

2.1 SLOs definition

We will define an SLO as the probability (P) that its associated SLI is within a specified (i.e., desirable) range. In such a case, we will say that the SLO is accomplished or fulfilled. Otherwise, it is violated. Mathematically, it can be expressed as in Eq. (1):

$$SLO = P(x \le sli \le y) \mid x \le y\,; \;\; \forall x, y \in SLI \tag{1}$$

Where x is the lower-bound and y is the upper-bound for the SLI, which is interpreted as the set of values that the metric can take. Latency, for example, assumes values in $\mathbb{R}^{+}$, because time will always be a positive real number. Hence, sli describes a specific value of the SLI for a given time. Notice that we have omitted any time reference in the previous equation, but given that sli is time-dependent while monitoring the SLO compliance state, we will obtain a time series. In general, SLO-based management of DCCS implies adapting the system to maximize the probability that the sli is within its range.

To define an SLO, we need to specify its operation range, "$y - x$". This means defining where we center the range and what its length is. For the following discussion we assume that the SLO range is normalized, so it is fair to compare ranges between different SLOs. Intuitively, the center of the range is the expected value for the SLI. Interestingly, this might vary depending on the device in which the service will be deployed. Simply put, the expected response time of a service performing a machine learning inference task in the Cloud will be lower than the same service in the Edge. The "length" of this range describes the criticality of the service. Services with large SLO ranges can adapt easily to many situations and, consequently, are less critical. Conversely, a short range indicates that the SLO is heavily constrained and the service is critical. Hence, it is important to consider the SLO range when elasticity strategies need to be applied. SLOs with longer ranges will be adapted with less effort; however, adapting the ones with shorter ranges will be safer, given that the propagation effects will have a lower impact on SLOs with longer ranges.

Running example. Hence, for each service requirement, an SLO must be defined. With the medical example, one could constrain the inference service processing time (T) between T_{min} and T_{max}. Of course, the lower bound for the processing time could be removed, but if there is knowledge about the service behavior, the lower bound can help identify faults or anomalies. When the underlying infrastructure is considered, i.e., the specific type of hardware that shall host the services, the SLOs definition might vary, and some other SLOs might

be required. For instance, considering that the gathering and pre-processing service is hosted in an SBC on the user. This might need a constraint on the device's power consumption (e.g., hourly average consumption < 8W), which can be reflected in the device's processing availability. Also, if the inference service is in an Edge device for privacy enhancement and latency minimization, this can shift the SLO on processing time (now being T'_{min} and T'_{max}).

2.2 Types of SLOs

SLOs are linked to services and specify their expected behavior. However, we must remember that services are components of a larger application. In that regard, the application or significant parts of it might need specific requirements. Imagine a machine learning pipeline (e.g., a set of sequential services that gathers data, pre-process, trains a model, and broadcasts it to edge devices) having specific time-based requirements for each service and an overall requirement of achieving a test accuracy of at least 95%. In this case, the accuracy SLO can be part of any of these services, but it is really an SLO for the pipeline meta-service[1]. In parallel, the underlying infrastructure belongs to other stakeholders. Hence, it is probable that they need to set requirements for their devices to ensure their performance when providing a host for several tenants. In such a case, a service hosted in the device will also have an SLO constraining, for instance, the total amount of CPU used. Interestingly, this case unveils an aspect of DeepSLOs, which is conflict analysis, i.e., the impossibility of simultaneously fulfilling two conflicting SLOs. Hence, a service SLO might need to be violated to fulfill the meta-service SLO. Section 4.3 will develop this aspect.

We can define three types of SLOs: infrastructure, service, and application (or meta-services). Figure 2 shows a graphical representation of each type of SLO. We will use the term high-level SLO for those related to the application or meta-services and low-level SLO for the ones that relate to the infrastructure components. Regarding the SLOs placement, services' SLOs will be located together with their service. For instance, both entities would share a pod in a Kubernetes-based application. Infrastructure SLOs would be deployed in services hosted in the critical infrastructure, i.e., devices that need to be specifically monitored to ensure their proper behavior. However, the placement of the application SLOs is not that clear. Actually, it brings a novel degree of freedom for system optimization. However, our intuition is that the system architecture will show the best candidate locations for these SLOs. It is necessary to minimize the overhead of system management and the resources in use. Hence, one can assume that proximity to data and to elasticity strategies is valuable. For clarification, by proximity to the elasticity strategies, we mean that there are no significant delays between the service that requires the elasticity strategy and the one that can apply.

Running example. The eHealth pipeline can include high-level SLOs that consider the overall cost of the application. Similarly, another high-level SLO can monitor the overall success of the alarm system. In contrast, lower-level SLOs are bond to the underlying infrastructure. Edge devices might have a GPU available for inference, but with shared usage, their overall usage throughout the day is limited. Or the SBCs might run on battery power limiting its availability.

[1]We use meta-service for services that could be grouped, e.g., a pipeline, and for services that provide functions beyond the application scope, e.g., an orchestrator.

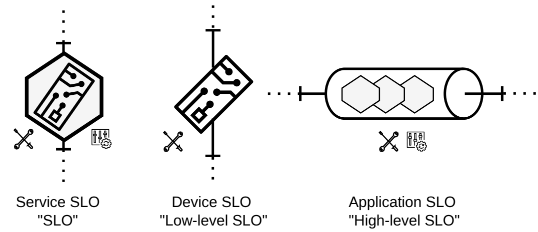

Figure 2: Types of SLO

2.3 Tailored adaptations

One crucial aspect that motivates defining SLOs, considering both the service and the device to be deployed, is the capacity to define tailored elastic strategies for service-device pair. An elastic strategy or adaptation is a change in the service or device that ensures the fulfillment of the SLO when it has been violated (reactive behavior) or when it is about to be violated (proactive behavior). In cloud-based applications, the adaptation capabilities are basically horizontal and vertical scaling; the Kubernetes autoscaler is the reference for both scientific and production systems. This dichotomy, horizontal or vertical scaling, is not always available at the Edge or in constraint devices. Hence, tailoring the elasticity strategies to the service-device pair is fundamental for autonomous and decentralized behavior.

Hence, when considering the service and the device, we are capable of knowing if the service will scale or if offloading to near and similar devices is an option [27]. Further, some devices can modify their characteristics. As an example, most NVIDIA Jetson devices[2] allow runtime configuration of their maximum energy consumption, or they can enable/disable the GPU at runtime. Hence, these configuration options are cast as elasticity strategies. Further, we consider service-based elasticity measures, which go beyond adding or removing more replicas. Services configuration can be changed if they have the proper interfaces to adapt their behavior for the specific moment. It is important to clarify that these measures do not aim to change the service logic but to change its behavior. For instance, one can easily change the granularity of the input data to alleviate data processing tasks [26]. Similarly, the ML model for inference processes can be selected by trading off accuracy with energy consumption. Hence, services must provide interfaces to trade-off characteristics regarding quality, cost, or performance.

Running example. Executing the inference service at an Edge device with GPU allows the design of two elasticity strategies that go beyond scaling, and might be specific for this type of host. For instance, if the GPU can be utilized on demand, the service can run initially without, and when the required number of inferences increases, the GPU is switched on. Further, if the GPU usage limit is reached, the service can be offloaded (or the requests re-directed) to another Edge device with GPU time available for the application.

[2]https://developer.nvidia.com/embedded/jetson-modules

3 SLO AREA OF INTEREST

Up to this point, we have been explaining that (1) SLOs have to define service requirements together with the hosting device, that (2) we have different types of SLOs, and that (3) tailored adaptations for the service-device pair are needed to ensure SLO compliance. However, we have skipped a crucial aspect: decentralization.

Decentralization requires that each SLO knows its needs and capacities as any autonomous agent. We define the area of interest of an SLO as those variables and parameters that the SLO must consider to evaluate its current state autonomously and to take action accordingly. Hence, evaluation and adaptation are performed locally. Indeed, higher-level SLOs will require data coming from different parts of the system. Hence, their area of interest might be more extensive. Regardless, filtering out redundant or irrelevant data for the specific SLO is fundamental to achieving scalable system management. Interestingly, why would someone need to check influencing variables or parameters to the SLO if directly observing the SLO is feasible? The answer is twofold; on one side, having the knowledge of the influencing variables provides information on the causes of the SLO behavior, which leads to explainable and accountable systems. On the other side, the influencing variables and parameters are needed to properly take the most adequate elasticity strategy. Hence, providing an area of interest per SLO maximizes the level of decentralization for any SLO-based system. Local decisions are framed to an SLO. Therefore, its elastic strategies [39] are cast as parameters that the SLO can consider to choose its available elasticity strategies. Hence, the possible adaptations for the SLO are known and available only locally.

Running example. Let's assume that there is at least 1 SLO defined per service, at least 1 higher-level SLO, and another lower-level one. At this point, besides monitoring the SLO behavior, other metrics and parameters of the system are required to be tracked. This consists of metrics of the underlying infrastructure, from CPU/GPU usage to power consumption or requests received per second; metrics derived from the services such as inference time or pre-processing data queue; and finally, parameters that can influence the service behavior and might be used as elasticity strategies, this can range from ML model, data pre-processing steps, or GPU status (on/off) at the Edge node. Expert knowledge is needed to build this list of candidate elements to track. However, once DCCS are more common, it will be easier to define this step; further, ML technologies can help suggest required metrics or parameters.

3.1 Markov Blanket

The Markov Blanket[3] is the mathematical concept that defines each SLO area of interest. This concept has two valuable perspectives. On one side, the Markov Blanket, defined by J. Pearl [18], is purely probabilistic. Conversely, the Markov Blanket used by K. Friston [8] to define the Free Energy Principle has an ontological perspective, i.e., it is used to define what any thing is. The Markov Blanket of a random variable, x, (in the probabilistic sense of the meaning) contains all those variables that make x conditionally independent of any other set of variables. In a Bayesian Network, the Markov Blanket of a variable can be visually identified because it is always composed of its parents, children, and co-parents. Formally, if $MB(x)$ are the variables from the Markov Blanket of x and Y are all other variables, then the following equation holds:

$$P(x|MB(x), Y) = P(x|MB(x)) \tag{2}$$

It is possible to bring this concept to the DCCS by assuming that this central variable, x, is the SLO at hand. Then, applying the Markov Blanket over this variable provides the set of variables that will affect the behavior of the SLO compliance. Hence, this sets a causality filter over the SLO, identifying only the system metrics that have to be tracked for the SLO, heavily reducing the time needed to assess the SLO status and inferring possible adaptation measures, see Sedlak et al. [27] figure 8 for an initial quantitative result of the reduction. Indeed, this minimizes the monitoring effort while maximizing its effectiveness.

Discovering the Markov Blanket of an SLO is a complex task, and when considering our previous work, it requires the combination of two types of knowledge. On the one hand, there is a need for expert knowledge to identify the system variables that might be needed to assess the SLO. On the other hand, it requires the system's data to use Markov Blanket discovery methods to quantify the relation of the selected variables with the SLO, such as the one presented by Fu et al. [10], the interested reader can check this survey for more detail and candidate methods [36].

The second perspective of the Markov Blanket uses the concept to build the interfaces of a thing (the SLO) with its environment. It defines the sensory states, those variables affected by the environment that influence the internal state (SLO compliance). And the active states are those variables affected by the internal state that influence the environment. We leverage this dichotomy to identify those variables that the SLO can change to affect its behavior, which we generally call parameters. It is needed to stop here for clarification: the SLO describes the state of a system component. Further, we assume that if components were completely isolated, the SLO would not change its state. Hence, it is the environment that influences the SLO to deviate from its equilibrium. The environment is composed of many things (all external to the SLO). It includes other application services, the users, the hardware in which it is hosted (assuming a multi-tenant environment), etc. When defining the active states, we have said that they are those states that influence the environment. This is partially true. What they do is affect the relation of the SLO with the environment. This means that if something in the environment is making the SLO deviate from its equilibrium, the SLO (meaning the autonomous agent controlling it) has to perform an elasticity strategy to revert that trend. This can be on the environment itself (external action), e.g., spawning a new service instance to absorb the high demand. Still, it can also change the service itself (internal action) [25], e.g., reduce the granularity of the data being analyzed, adapting the QoS offered to keep the relation between the SLO and the environment in equilibrium.

Figure 3 represents an SLO with its Markov Blanket: M1, M2, and M3 depict metrics influencing the SLO behavior, while M4 is a metric that does not influence the SLO. Additionally, the two squares at the bottom represent action states, which can influence the relation of the SLO with its environment. The next section will

[3]Formally, there is a difference between the Markov Blanket and the Markov Boundary. The latter is the minimal set of the first. However, to align with previous work and because this distinction is not critical for our work, we indiscriminately use the term Markov Blanket.

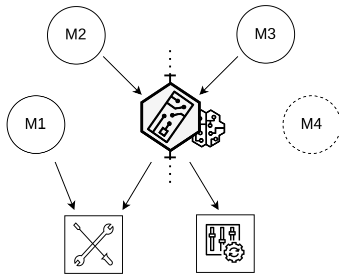

Figure 3: Markov Blanket representation of an SLO.

introduce the brain behind the SLO, i.e., how we provide autonomy to the SLO.

Running example. *Now that the metrics and parameters available for each SLO are defined, it is necessary to keep only those directly affecting the SLO at hand. Computing the Markov Blanket of each SLO allows for determining which variables (i.e., the metrics and parameters defined previously) the SLO is conditionally dependent on. This way, all those variables that do not directly affect the SLO can be discarded, reducing the total amount of data to be analyzed. The requirement is modeled as a simplified Bayesian network, where the SLO is the central variable. Further, those parameters that might be used as elasticity strategies are linked with other system metrics, which will help identify the best strategy according to the system status. For instance, the inference processing time will have dependencies only with the GPU status and the model time. Hence, if the GPU status is already ON, the only way to reduce the processing time is by using a cached model that requires less computational effort.*

3.2 Autonomy

At this point, SLOs have all but one required ingredient to behave autonomously: intelligence. Simply put, the capacity of the SLO/service to autonomously decide how to adapt given its current state. Currently, software systems have three main directions to achieve intelligence: rule-based, model-based, or agent-based. Firstly, dealing with large, heterogeneous, and distributed systems precludes the usage of rule-based decisions as the space of possible situations is too large and complex for anyone to predetermine all rules. Interestingly, this is the standard approach for state-of-the-art Cloud systems (i.e., the Kubernetes autoscaler). This works due to Cloud homogeneity and centralization.

However, research is already going beyond this when considering the Edge. For example, the work of Toka et al. [34] develops AI-based models to manage Edge resources. Further, model-based requires previously specifying the model by using the underlying laws of physics of the system or its data to build the model. For instance, Liang et al. [14] build models for different Edge devices that perform machine learning tasks using queue theory. However, this results in developing and validating specific models for each service and device type combination. Model-based approaches using deep learning have also great success. However, the amount of data to train these models is huge; for instance, Jeong et al. [12] used 30 days of data for training, while the generalization capabilities on dynamic environments, such as in the CC, still need to be proven.

Lastly, agent-based systems can learn a behavioral model progressively while performing actions. As explained by J. Pearl [19], performing actions on the systems incorporates information to the data about the system's behavior that can't be seen only from observational data. The most common agent-based intelligence is the one brought by reinforcement learning. Specifically, most of its applications for Edge systems are model-free, which means that the consequences of actions are not evaluated before the action is taken. Formally, the probability of the new state (s') given the current state (s) and the action taken (a) is not assessed ($P(s'|s, a)$). See the works of Xiong et al. [38] for task allocation or the work from Tang and Wong [33] for task offloading. In any case, agent-based techniques usually require time to learn properly and suffer from the exploitation-exploration trade-off.

Hence, we opt for using the combination of both, agent-based intelligence with a system model. Also, instead of using reinforcement learning, we use active inference, which derives from the Free Energy Principle; refer to works from K. Friston [7] or a recent work from R. Smith et al. [30] for a comprehensive explanation. Active inference is an agent-based solution and is convenient to make SLOs intelligent for the following reasons. First, agent-based solutions are well-suited for decentralized systems, where components can have their own autonomous agent, and they are self-adaptive. Second, the formulation of active inference is perfectly aligned with the Markov Blanket representation of the system. This enables a complete and more straightforward integration into the SLO-based model that we are proposing. Third, as it is derived from the Free Energy Principle, its objective function is not reward-based but to improve its model of the environment [22]. This subtle difference better suits systems that might need to change their requirements with time, given that when the requirement changes and the model no longer fits, it will always try to make the model match with the observations. Fourth, active inference allows injecting the expected observations into the model, i.e., fulfilling its SLO becomes the main driver for the agent actions. Hence, it can learn how their actions affect the SLO fulfillment.

As previously mentioned, the exploitation-exploration trade-off is always challenging for agent-based intelligent systems. In that regard, we can quantify the risk of new observation balanced against the information gain (e.g., model improvement) that it can provide, and depending on the criticality of the SLO (i.e., the length of its operation range), we can weigh the risk against the gain.

Running example. *Now that each SLO is modeled as the central node of a Markov Blanket, we can use active inference to improve the SLO behavioral model and to decide the best policy to keep the SLOs fulfilled. Improving the model means closing the gap between the predicted outcomes of using elasticity strategies with respect to the actual behavior of the system. Simply put, what I believe will happen if I turn on the GPU against what really happens when I do it. In that regard, active inference can build policies that balance the learning of the model (when the system behavior is stable) with the SLO stability optimization. Taking back the running example, the effect of GPU usage on the processing time is favorable. However, the model can be refined so that the effect can be quantitatively measured, and hence, when the GPU capacity is exceeded, another elasticity strategy can be used, e.g., changing the ML model.*

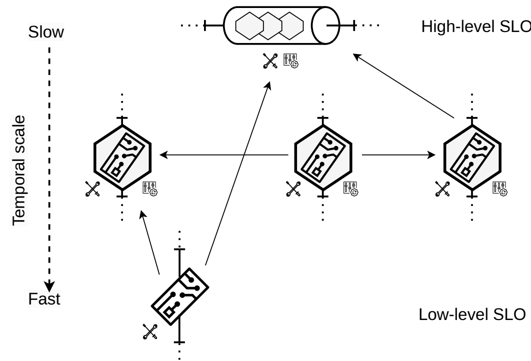

Figure 4: A DeepSLO representation with a high-level SLO, three SLOs and one low-level SLOs. Dependencies between SLOs are depicted with arrows.

4 DeepSLO

We have described how to define SLO-based requirements for DCCS considering the service-device pair and explained how to decentralize and make each SLO autonomous using the Markov Blanket concept and active inference. In that regard, from the previous section, we have a behavioral model of each SLO that we use to understand its performance and plan future elasticity strategies if the SLO is violated or about to be violated.

However, we must remember that DCCS are complex and interconnected systems. Hence, we need to account for these characteristics, and when doing so, the DeepSLO emerges. Figure 4 shows a representation for a DeepSLO with three levels. DeepSLOs are hierarchical structures that link all SLOs of a single system. In that regard, one can imagine it as the composition of all SLOs defined for a system. Further, this is a hierarchical structure, where higher-level SLOs are above the hierarchy and lower-level SLOs are below. This structure appears because high-level SLOs will require input from lower-level ones, generating different temporal scales within the DeepSLO structure.

Running example. *So far, we have defined SLOs for each service, the overall ML pipeline, and some critical infrastructure elements. These SLOs have been modeled with a Markov Blanket and provided with autonomy by active inference. Now, we are building a model of the entire application, which connects the SLOs if they are conditionally dependent (using Bayesian network learning techniques), this way we can see if there are SLOs that are not compatible or elasticity strategies that might have a negative effect on a connected SLO. Hence, the DeepSLO will allow the application stakeholders to address design and runtime trade-offs properly. For instance, take the two overall SLOs, cost (infrastructure) and user satisfaction (QoE), and imagine that the inference service requires improving its processing time; it has 2 options: switching ON the GPU or changing the ML model. Hence, we can easily foresee that these will have an effect on the high-level SLOs. Hence, understanding the propagation effects from the elasticity strategies and the SLO violations on the other system components is vital to properly deciding the best set of actions at each given moment while making all stakeholders aware.*

4.1 SLOs Links

The DeepSLO structure requires bridging all Markov Blanket structures to generate a Bayesian Network (BN). This holistic structure defines the dependencies between the system requirements. Before, with the MB representation of an SLO, we could describe the behavior of a single requirement; now, we hide the internal behavior of the requirement to focus on the dependency between requirements. The edges on the BN represent these dependencies, stemming from three different sources.

The first set of edges that build the DeepSLO come from services' dependencies. DCCS applications are a set of interconnected services. Hence, the dependencies between these services are added as edges between Markov Blankets. Of course, these dependencies are going to be observed in the data, but knowing the application architecture and dependencies allows us to add a set of initial constraints on the methodology to build the DeepSLO structure.

The second set of edges emerges from the data. The system's data can be analyzed with techniques that build Bayesian networks. Consider the work from Scurati et al. [24] or the survey from Scanagatta et al. [23] for a comprehensive understanding of the available techniques. Hence, the DeepSLO structure can add the dependencies between hardware and network metrics shared between different Markov Blankets. These dependencies stem from shared devices or geographical distances. Further, the constraints specified by the services can finally be instantiated in this step when the system's data is analyzed, and hence, service dependencies are connected through specific system variables. An important point to consider here is semantics. It is crucial to know whether these are the same variables (e.g., they have the same names, they statistically show the same pattern, etc.) to link them from different Markov Blankets. Otherwise, it is likely to omit relevant dependencies.

The final set of edges might not appear in the data – it is related to high-level SLOs that specify requirements in terms of overall system performance or cost. Hence, these edges need to be added manually. Hence, the individual models of the SLOs (represented as Markov blankets) have to incorporate variables accounting for these relations. As an example, imagine that we have a high-level SLO constraining the amount of energy that the system can use. Then, we will need to incorporate into the individual Markov Blankets the knowledge of the energy that service is consuming and, if possible, means to change that value. This is a simple example, but in general, business or application-related dependencies can be very hard to find in data, and they might need explicit modeling.

We are providing a bottom-up description of the DeepSLO, and hence, now we need to express the need to complete each individual SLO model to account for the system dependencies. However, when designing a system, its high-level SLOs (top-level requirements) are known. Hence, these variables and parameters linking individual behaviors can be considered as part of each Markov Blanket model from the beginning.

Running example. *When building the DeepSLO structure from the system's data, there will be variables that, if they are not properly included on the system's analysis, the links will not appear. Simply put, if the cost of switching ON a GPU is not considered, the SLO cost will never be affected. In that regard, if we are aware that there are links that come from the services dependencies, others that stem*

from the infrastructure usage, and others that depend on the higher level SLOs, making sure that all required data is considered becomes simpler.

4.2 Network overlay

The DeepSLO is a geographically distributed structure to manage DCCS in a decentralized manner. Hence, keeping the structure functional during the application life-cycle requires building an overlay network on top of the one for the application itself. This network has to allow information propagation through the network, making all interested SLOs aware of any change in the system. As explained in Section 3, leveraging the Markov Blankets structure to evaluate SLOs minimizes the sources of information that need to be gathered. Further, the connections between Markov Blankets that arose from the system's data do not require specific connections as each Markov Blanket independently will be aware of the metric's change, as they will be subscribed by definition. Hence, they just need the means to monitor.

Hence, there is a need to manage connections for the edges derived from services' dependencies and those that do not appear in the data and relate to high-level SLOs. Interestingly, the information that relates to services' dependencies can also be obtained by leveraging the already existing connection between the services. Simply put, if two services within a pipeline depend on throughput or latency, this information can be obtained from the messages that these services exchange within the application logic, alleviating the need to build an extra connection to convey this information. Further, suppose the information they have to exchange is more complex, e.g., it can't be derived from the messages' headers. In that case, it can be envisioned adding encoders/decoders [28] at each end of the service in order to merge the system's information with the service's information that they usually share. This would be equivalent to using semantic communication among services [35].

The last type of connection relates to high-level SLOs and it originates from the third type of edges explained in Section 4.1. Connecting these high-level SLOs would require putting in place specific connections in the system to transfer the required information from lower-level to higher-level SLOs. In that regard, the benefit of being from lower levels to higher ones is that the higher-level SLOs have lower managing frequencies. Hence, the latency constraints for this information will be lower than the general latency required for low-level SLOs of the system. In that regard, strategies to group or simplify the data that is required to be sent can be put in place to minimize the impact of these connections on the system's overall cost.

Running example. *The DeepSLO structure forces to share data between SLOs, and hence, the question is how to minimize its overhead. In that regard, we use the previous classification of dependencies to see how these connections can be built. In some situations, the information might be embedded in the services data exchange, but others will need to be built ad-hoc. Hence, deciding where the SLO is executed and orchestrated can help minimize this communication overhead.*

4.3 Prioritization and conflict resolution

Conflicting SLOs refer to different SLOs that cannot be fulfilled simultaneously, or in other words, the fulfillment of one makes the other fail. It is important to clarify that we do not foresee SLOs conflicting within a single service, even if more than one SLO specifies the service. We assume that in design time for a single service the SLOs will have been designed to properly address the required trade-offs.

However, high-level SLOs, the ones that specify meta-services, application or business requirements can conflict with lower-level SLOs, i.e., the ones specifying single services or hardware. This is exacerbated by the fact that the stakeholders supporting high-level SLOs will, most likely, be different than the ones supporting the low-level ones. In that regard, special care has to be taken at design time in order to understand and incorporate all DCCS requirements.

Indeed, the best solution is a design that does not allow this type of conflict. In that regard, we opt for using top-bottom diffusion and bottom-up capillarity techniques to specify the requirements. Top-bottom diffusion consists of specifying only the higher-level SLOs of the system and study which are the equilibrium values for the low-level SLOs. On the contrary, bottom-up capillarity implies specifying only the low-level SLOs and check which are the equilibrium values for the high-level SLOs. If both methodologies are taken, it might be possible to find those trade-offs between values that account for the fulfillment of all SLOs.

Regardless, if it is not possible to fulfill all SLOs, our intuition accounts for prioritizing high-level SLOs over low-level ones, i.e., using the hierarchical structure of the DeepSLO to define the prioritization between SLOs. However, we encourage all stakeholders to properly address this issue at design time to foster the best solution for the application being developed.

Running example. *Let's take back the basic case of the conflicting SLOs, where cost and QoE conflicted. Considering the use case that we have been presenting, the eHealth, one can argue that health is a priority, and hence, QoE, i.e., the proper response to a medical situation, is a priority over the cost (perhaps the system will send a higher invoice to the user). The point here is that in most situations, this type of conflict requires that the stakeholders agree on a prioritization of the requirements. Hence, building the overall application model as a Bayesian network provides clarity on the decisions that are required and the effects that will be produced, which is a fundamental feature when several stakeholders need to agree.*

5 CONCLUSIONS

DeepSLOs are a fundamental step to developing DCCS with explainable behaviors, which self-adapt to changes in external conditions while fulfilling their requirements. The DeepSLO is an artifact that specifies the complete behavior of a DCCS for decentralized and autonomous management while accounting for key general behaviors of the system. This article provides a bottom-up description of the DeepSLO artifact. Starting with the breakthrough of DCCS with respect to Cloud systems, we pinpoint the needs of DCCS and then, step-by-step, build the functionalities needed to fulfill them. Table 1 presents a summary of the functions required to build DeepSLOs with potential methods to achieve them. Further, several have been tested in the context of SLO management for constraint and heterogeneous devices with promising results.

Indeed, our vision is to develop a sound and formal methodology to build systems for the CC, and we expect to attach powerful

Table 1: Summary of methods for DeepSLO-related functions

Function	Potential methods
Services characterization	Profiling as in [15]
Devices characterization	Feature space mapping as in [20]
Latency SLO	Deep learning [37]
SLO definition	Design process
Tailored adaptations	For device and service as in [26]
SLO area of interest	Markov Blanket discovery as in [10]
SLO Autonomy	Active inference as in [27]
DeepSLO links	Design process
DeepSLO network overlay	Semantic communication as in [28]
DeepSLO management	Bayesian inference

mathematical tools to it so that the future applications of the CC are helpful, sustainable, and resilient. In that regard, besides refining and expanding the applicability of the methods that we have tested in previous works, we have several research directions that we aim to push. As an example, we look forward to modeling the relations of a DeepSLO as differential equations, for instance, using Dynamic causal modeling [9] to address the diffusion and capillarity methods with two sets of initial conditions. This would lead to an automatic definition of SLO ranges. Further, we are aware that the definition of service-device requirements for DCCS can be a complex task. There is a myriad of different services and devices; hence, having adequate knowledge of all service-device combinations is a gargantuan task. Regardless, once data is available, a deep learning approach can simplify its development, leaving only a verification step required. It is still part of our research agenda to identify and apply methods that can ease and, if possible, automate this process. We envision using large-language models (LLMs) to acquire the expert knowledge to obtain the set of candidate system metrics and the usage of graph-neural networks (GNNs) to build initial DeepSLOs from the input of an LLM automatically.

ACKNOWLEDGEMENT

Funded by *European Union (TEADAL, 101070186)*. Views and opinions expressed are those of the authors and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible.

REFERENCES

[1] Hussain AlJahdali, Abdulaziz Albatli, Peter Garraghan, Paul Townend, Lydia Lau, and Jie Xu. 2014. Multi-tenancy in Cloud Computing. In *2014 IEEE 8th International Symposium on Service Oriented System Engineering*. 344–351. https://doi.org/10.1109/SOSE.2014.50

[2] Ahmad Alzu'bi, Ala'a Alomar, Shahed Alkhaza'leh, Abdelrahman Abuarqoub, and Mohammad Hammoudeh. 2024. A Review of Privacy and Security of Edge Computing in Smart Healthcare Systems: Issues, Challenges, and Research Directions. *Tsinghua Science and Technology* 29, 4 (Aug. 2024), 1152–1180. https://doi.org/10.26599/TST.2023.9010080

[3] Himani Bajaj, Anjali Sharma, Deepshi Arora, Mayank Yadav, Devkant Sharma, and Prabhjot Singh Bajwa. 2024. Challenges in E-Waste Management. In *Sustainable Management of Electronic Waste*. John Wiley & Sons, Ltd, 201–220. https://onlinelibrary.wiley.com/doi/abs/10.1002/9781394166923.ch10

[4] Pengfei Chen, Yong Qi, and Di Hou. 2019. CauseInfer: Automated End-to-End Performance Diagnosis with Hierarchical Causality Graph in Cloud Environment. *IEEE Transactions on Services Computing* 12, 2 (March 2019), 214–230. https://doi.org/10.1109/TSC.2016.2607739

[5] Can Cui, Yunsheng Ma, Xu Cao, Wenqian Ye, and Ziran Wang. 2024. Drive As You Speak: Enabling Human-Like Interaction With Large Language Models in Autonomous Vehicles. 902–909.

[6] Schahram Dustdar, Victor Casamayor Pujol, and Praveen Kumar Donta. 2023. On Distributed Computing Continuum Systems. *IEEE Transactions on Knowledge and Data Engineering* 35, 4 (April 2023), 4092–4105. https://doi.org/10.1109/TKDE.2022.3142856

[7] Karl Friston, Thomas FitzGerald, Francesco Rigoli, Philipp Schwartenbeck, John O'Doherty, and Giovanni Pezzulo. 2016. Active inference and learning. *Neuroscience & Biobehavioral Reviews* 68 (Sept. 2016), 862–879. https://doi.org/10.1016/J.NEUBIOREV.2016.06.022

[8] Karl Friston, James Kilner, and Lee Harrison. 2006. A free energy principle for the brain. *Journal of Physiology Paris* 100, 1-3 (July 2006), 70–87. https://doi.org/10.1016/j.jphysparis.2006.10.001

[9] K. J. Friston, L. Harrison, and W. Penny. 2003. Dynamic causal modelling. *NeuroImage* 19, 4 (Aug. 2003), 1273–1302. https://doi.org/10.1016/S1053-8119(03)00202-7

[10] Shunkai Fu and Michel C. Desmarais. 2008. Fast Markov Blanket Discovery Algorithm Via Local Learning within Single Pass. In *Advances in Artificial Intelligence*. Springer, Berlin, Heidelberg, 96–107. https://doi.org/10.1007/978-3-540-68825-9_10

[11] Noor Ul Huda, Ijaz Ahmed, Muhammad Adnan, Mansoor Ali, and Faisal Naeem. 2024. Experts and intelligent systems for smart homes' Transformation to Sustainable Smart Cities: A comprehensive review. *Expert Systems with Applications* 238 (March 2024), 122380. https://doi.org/10.1016/j.eswa.2023.122380

[12] Byeonghui Jeong, Seungyeon Baek, Sihyun Park, Jueun Jeon, and Young-Sik Jeong. 2023. Stable and efficient resource management using deep neural network on cloud computing. *Neurocomputing* 521 (Feb. 2023), 99–112. https://doi.org/10.1016/j.neucom.2022.11.089

[13] Faria Kalim. 2020. *Satisfying service level objectives in stream processing systems*. Ph. D. Dissertation.

[14] Qianlin Liang, Walid A. Hanafy, Ahmed Ali-Eldin, and Prashant Shenoy. 2023. Model-driven Cluster Resource Management for AI Workloads in Edge Clouds. *ACM Transactions on Autonomous and Adaptive Systems* 18, 1 (March 2023), 2:1–2:26. https://doi.org/10.1145/3582080

[15] Andrea Morichetta, Víctor Casamayor Pujol, Stefan Nastic, Schahram Dustdar, Deepak Vij, Ying Xiong, and Zhaobo Zhang. 2023. PolarisProfiler: A Novel Metadata-Based Profiling Approach for Optimizing Resource Management in the Edge-Cloud Continnum. In *2023 IEEE International Conference on Service-Oriented System Engineering (SOSE)*. 27–36. https://doi.org/10.1109/SOSE58276.2023.00010

[16] Andrea Morichetta, Thomas Pusztai, Deepak Vij, Víctor Casamayor Pujol, Philipp Raith, Ying Xiong, Stefan Nastic, Schahram Dustdar, and Zhaobo Zhang. 2023. Demystifying deep learning in predictive monitoring for cloud-native SLOs. In *2023 IEEE 16th International Conference on Cloud Computing (CLOUD)*. 1–11. https://doi.org/10.1109/CLOUD60044.2023.00013

[17] Michael R. Nelson. 2009. Building an Open Cloud. *Science* 324, 5935 (June 2009), 1656–1657. https://doi.org/10.1126/science.1174225

[18] Judea Pearl. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[19] Judea Pearl and Dana Mackenzie. 2018. *The Book of Why: The New Science of Cause and Effect*. Basic Books, Inc., USA.

[20] Victor Casamayor Pujol, Andrea Morichetta, and Stefan Nastic. 2023. Intelligent Sampling: A Novel Approach to Optimize Workload Scheduling in Large-Scale Heterogeneous Computing Continuum. In *2023 IEEE International Conference on Service-Oriented System Engineering (SOSE)*. 140–149. https://doi.org/10.1109/SOSE58276.2023.00024

[21] Haoran Qiu, Subho S. Banerjee, Saurabh Jha, Zbigniew T. Kalbarczyk, and Ravishankar K. Iyer. 2020. FIRM: an intelligent fine-grained resource management framework for SLO-oriented microservices. In *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation (OSDI'20)*. USENIX Association, USA, 805–825.

[22] Noor Sajid, Philip J. Ball, Thomas Parr, and Karl J. Friston. 2021. Active Inference: Demystified and Compared. *Neural Computation* 33, 3 (March 2021), 674–712. https://doi.org/10.1162/NECO_A_01357

[23] Mauro Scanagatta, Antonio Salmerón, and Fabio Stella. 2019. A survey on Bayesian network structure learning from data. *Progress in Artificial Intelligence* 8, 4 (Dec. 2019), 425–439. https://doi.org/10.1007/S13748-019-00194-Y/TABLES/1

[24] Marco Scutari, Catharina Elisabeth Graafland, and José Manuel Gutiérrez. 2019. Who learns better Bayesian network structures: Accuracy and speed of structure learning algorithms. *International Journal of Approximate Reasoning* 115 (Dec. 2019), 235–253. https://doi.org/10.1016/J.IJAR.2019.10.003

[25] Boris Sedlak, Victor Casamayor Pujol, Praveen Kumar Donta, and Schahram Dustdar. 2023. Controlling Data Gravity and Data Friction: From Metrics to Multidimensional Elasticity Strategies. In *2023 IEEE International Conference on Software Services Engineering (SSE)*. 43–49. https://doi.org/10.1109/SSE60056.2023.00017

[26] Boris Sedlak, Victor Casamayor Pujol, Praveen Kumar Donta, and Schahram Dustdar. 2023. Designing Reconfigurable Intelligent Systems with Markov Blankets. In *Service-Oriented Computing (Lecture Notes in Computer Science)*, Flavia Monti, Stefanie Rinderle-Ma, Antonio Ruiz Cortés, Zibin Zheng, and Massimo Mecella (Eds.). Springer Nature Switzerland, Cham, 42–50. https://doi.org/10.1007/978-3-031-48421-6_4

[27] Boris Sedlak, Victor Casamayor Pujol, Praveen Kumar Donta, and Schahram Dustdar. 2023. Equilibrium in the Computing Continuum through Active Inference. https://doi.org/10.48550/arXiv.2311.16769

[28] Hyowoon Seo, Jihong Park, Mehdi Bennis, and Mérouane Debbah. 2021. Semantics-Native Communication with Contextual Reasoning. (Aug. 2021). https://doi.org/10.48550/arxiv.2108.05681

[29] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. 2016. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal* 3, 5 (Oct. 2016), 637–646. https://doi.org/10.1109/JIOT.2016.2579198

[30] Ryan Smith, Karl J Friston, and Christopher J Whyte. 2022. A step-by-step tutorial on active inference and its application to empirical data. *Journal of Mathematical Psychology* 107 (2022), 102632. https://doi.org/10.1016/j.jmp.2021.102632

[31] Ion Stoica and Scott Shenker. 2021. From cloud computing to sky computing. In *Proceedings of the Workshop on Hot Topics in Operating Systems (HotOS '21)*. Association for Computing Machinery, New York, NY, USA, 26–32. https://doi.org/10.1145/3458336.3465301

[32] Ajaya K Swain and Valeria R Garza. 2023. Key factors in achieving Service Level Agreements (SLA) for Information Technology (IT) incident resolution. *Information Systems Frontiers* 25, 2 (2023), 819–834.

[33] Ming Tang and Vincent W.S. Wong. 2022. Deep Reinforcement Learning for Task Offloading in Mobile Edge Computing Systems. *IEEE Transactions on Mobile Computing* 21, 6 (June 2022), 1985–1997. https://doi.org/10.1109/TMC.2020.3036871

[34] László Toka, Gergely Dobreff, Balázs Fodor, and Balázs Sonkoly. 2021. Machine Learning-Based Scaling Management for Kubernetes Edge Clusters. *IEEE Transactions on Network and Service Management* 18, 1 (March 2021), 958–972. https://doi.org/10.1109/TNSM.2021.3052837

[35] Elif Uysal, Onur Kaya, Anthony Ephremides, James Gross, Marian Codreanu, Petar Popovski, Mohamad Assaad, Gianluigi Liva, Andrea Munari, Touraj Soleymani, Beatriz Soret, and Karl Henrik Johansson. 2021. Semantic Communications in Networked Systems: A Data Significance Perspective. *arXiv* (March 2021). https://doi.org/10.48550/arxiv.2103.05391

[36] Matthew J. Vowels, Necati Cihan Camgoz, and Richard Bowden. 2021. D'ya like DAGs? A Survey on Structure Learning and Causal Discovery. (March 2021). https://arxiv.org/abs/2103.02582v2

[37] Jing Wu, Lin Wang, Qirui Jin, and Fangming Liu. 2024. GRAFT: Efficient inference serving for hybrid deep learning with SLO guarantees via DNN re-alignment. *IEEE Transactions on Parallel and Distributed Systems* 35, 2 (2024), 280–296.

[38] Xiong Xiong, Kan Zheng, Lei Lei, and Lu Hou. 2020. Resource Allocation Based on Deep Reinforcement Learning in IoT Edge Computing. *IEEE Journal on Selected Areas in Communications* 38, 6 (June 2020), 1133–1146. https://doi.org/10.1109/JSAC.2020.2986615

[39] Fan Zhang, Xuxin Tang, Xiu Li, Samee U Khan, and Zhijiang Li. 2019. Quantifying cloud elasticity with container-based autoscaling. *Future Generation Computer Systems* 98 (2019), 672–681. https://doi.org/10.1016/j.future.2018.09.009

[40] Zhizhou Zhang, Murali Krishna Ramanathan, Prithvi Raj, Abhishek Parwal, Timothy Sherwood, and Milind Chabbi. 2022. {CRISP}: Critical Path Analysis of {Large-Scale} Microservice Architectures. 655–672. https://www.usenix.org/conference/atc22/presentation/zhang-zhizhou