Markov Blanket Composition of SLOs

Boris Sedlak, Victor Casamayor Pujol, Praveen Kumar Donta, and Schahram Dustdar

Distributed Systems Group, Vienna University of Technology (*TU Wien*), Vienna 1040, Austria.
{b.sedlak, v.casamayor, pdonta, dustdar}@dsg.tuwien.ac.at

2024 IEEE International Conference on Edge Computing and Communications (EDGE) | 979-8-3503-6849-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/EDGE62653.2024.00025

***Abstract*—Smart environments use composable microservices pipelines to process Internet of Things (IoT) data, where each service is dependent on the outcome of its predecessor. To ensure Quality of Service (QoS), individual services must fulfill Service Level Objectives (SLOs); however, SLO fulfillment is dependent on resources (e.g., processing or storage), which are scarcely available within the Edge. Hence, when distributing services over heterogeneous devices, this raises the question of where to deploy each service to best fulfill both its own SLOs as well as those imposed by dependent services. In this paper, we maximize SLO fulfillment of a pipeline-based application by analyzing these dependencies. To achieve this, services and hosting devices alike are extended with a Markov blanket (MB) – a probabilistic view into their internal processes – which are composed into one overarching model. Given a mutable set of services, hosts, and SLOs, the composed MB allows inferring the optimal assignment between services and edge devices. We evaluated our method for a smart city scenario, which assigned pipelined services (e.g., video processing) under constraints from subsequent services (e.g., consumer latency). The results showed how our method can support infrastructure providers by optimizing SLO fulfillment for arbitrary devices currently available.**

***Index Terms*—Service Level Objectives, Computing Continuum, Markov Blanket, Quality of Service, Bayesian Networks**

I. Introduction

Public spaces are increasingly covered with sensor networks, e.g., road surveillance or parking sensors, which are used to build compound services such as offered by smart cities [1] or smart homes. In particular, this can include microservice pipelines, where one service (e.g., road analysis) feeds its results to multiple other services (e.g., traffic routing). While data collection is often carried out through Internet of Things (IoT) devices, the tendency is to locate data processing services at nearby edge devices; this promises low latency and improved privacy. However, whereas the Cloud counted on vast amounts of virtualized scalable resources [2], the limited amount of Edge resources promoted the rise of the Computing Continuum (CC) [3], [4] – a coherent integration of multiple computational tiers, starting from the IoT, over Edge and Fog, up to the Cloud. Thus, from data provisioning, over processing, up to consumption, services can be allocated and scaled over this large-scale multi-tenant distributed system [5].

To ensure high-level requirements, such as availability or response time, the Cloud allows clients to specify Service Level Objectives (SLOs); by evaluating Service Level Identifiers (SLIs), e.g., system metrics, it is decidable if SLOs are fulfilled. To trace a system's behavior at finer granularity, high-level SLOs are diffused into smaller chunks [6], [7]; for compound services (e.g., pipelines), this boils down to requirements that individual services must fulfill. Within a pipeline, each service poses its own SLOs: a processing service, for example, might aim for efficiency, whereas a consumer service could aim for high video resolution, i.e., instances of Quality of Service (QoS) and Quality of Experience (QoE). Services, however, depend on the quality provided by their predecessor(s) and expected by successor(s); this constrains the actions of individual services because they must consider how local changes influence dependent services. Furthermore, SLO fulfillment is affected by hosting infrastructure [8], i.e., low-resource devices (e.g., Edge) might not be able to fulfill performance constraints to the same extent as high-resource devices (e.g., Fog/Cloud). Hence, the deployment of a service has strong implications not only for its own SLO fulfillment [9] but also for dependent services'.

To maximize SLO fulfillment throughout a pipeline, the infrastructure provider must know where to deploy each service – we call this an "assignment" between services and hosts. Estimating the quality of an assignment requires (1) an understanding of how dependent services constrain each other's requirements and (2) the implications for the service and the deployed host, i.e., hardware utilization and SLO fulfillment. However, existing works [10], [11] do not consider transitive dependencies (i.e., imposed by dependent services) nor estimate resource utilization per service. Without the latter, deploying multiple services on one device (i.e., multi-tenancy) is risky because it is uncertain whether multiple services can coexist with their respective resources [12]. Brute-force comparing all possible assignments empirically cannot be the solution: first, the underlying optimization problem is NP-hard [13], but secondly, even though you find the optimal assignment for one pipeline, the insight is not transferable to other scenarios. Services and hosts change over time due to demand and availability; hence, any solution should be able to repeatedly infer optimal assignments for changing setups.

In this paper, we present a 4-step methodology that finds the optimal service assignment by analyzing intersections between dependent services and their impact on hosting devices. In the first step, services and devices are extended with a Markov blanket (MB) – a probabilistic representation of their internal processes [14]. This allows predicting how changes to one variable (e.g., video resolution) change the conditional probabilities of another variable (e.g., throughput) [15]. To consider external factors, i.e., SLOs of dependent services, we propose the Markov Blanket Composition (MBC): a MBC comprises an entire pipeline and different processing hardware (i.e.,

979-8-3503-6849-9/24/$31.00 ©2024 IEEE
DOI 10.1109/EDGE62653.2024.00025

spread over the CC) in one coherent and modular structure.

Following our vision laid out in [3], [16], we use MBC to model hierarchical dependencies between services, in particular, how their actions and perceptions influence each other. Figure 1 shows how hierarchical services could take actions based on dependent services' observations. Given this MBC, we estimated how assigning one service to a particular device will impact its SLO and those of dependent services; by doing this repeatedly, the assignment with maximum SLO fulfillment can be inferred. Hence, the contributions of this article are:

1) The MBC as a mechanism for finding dependencies between services and their implications to processing hardware. Thus, services can incorporate and consider SLO fulfillment of dependent services and their hosts.
2) The generalization of services' resource utilization over heterogeneous hardware. This extrapolates the resource usage of a service at a particular device type and estimates the implications for comparable services or devices.
3) A collective inference mechanism that maximizes SLO fulfillment for a service pipeline by assigning services to CC devices. This considers dependencies in terms of QoS and QoE that services pose on their direct neighbors.

The remainder of the paper is structured as follows: Section II introduces background knowledge and an illustrative scenario, and Section III presents our methodology including MBC, which is implemented and evaluated in Section IV. Section V provides an overview of existing work in this field; finally, Section VI, concludes our paper with a future scope.

II. Preliminaries

This section provides an overview of concepts and definitions used throughout the paper; in particular, this involves existing tools and techniques required as background knowledge. Furthermore, this section provides an exemplary smart-city use case that will help to put these concepts into practice.

A. Concepts & Definitions

In the context of this paper, the most important entities are the (pipeline) services and hosting devices. We provide a formal representation of them, as well as their joint assignment.

Definition 1 (Service, s)**.** *A service is a utility offered to other (micro-)services or end users to fulfill a dedicated function; a service is described as* $s = \langle in, f, out, M_s, Q \rangle$*, where* $\underline{in}$ *and* $\underline{out}$ *are the data ingested and produced, respectively, and* $\underline{f}$ *is the operation on the data.* $\underline{Q}$ *is a list of SLOs that must be fulfilled during operation, and* $\underline{M_s}$ *contains a list of metrics observable during service execution.*

Microservice architectures (e.g., [17], [18]) form sequential processing pipelines by chaining together services. The implications between services are either known upfront or can be extracted with existing techniques [19]; the result is a dependency graph $K = (S, E)$, where directed edges (E) represent logical dependencies between services (S). Each dependency consists of a predecessor (p) and a successor (q). For two nodes $\{p, q\} \in S$, $pq \in E$ indicates that q is dependent on p. Also, q operates on results produced by p; hence, q is dependent on the time for executing f_p, and the network latency (nl) to transfer the result to q. The time for providing p's results (wt) to q is expressed as

$$wt(p, q, d) = time(f_p(d)) + nl_{p \to q} \tag{1}$$

where input data d is ingested to p. Any property of *in* and *out* (e.g., data size), as well as $time(f)$ and consequently wt, can be observed as part of M_s. Naturally, nl depends on the location (l) of host devices; however, notice that in the context of this paper, we assume nl to be independent of data size.

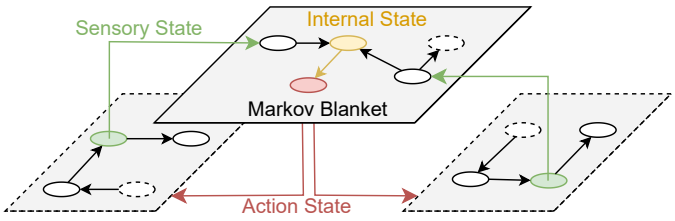


Fig. 1: Hierarchical dependencies between different Markov blankets

Definition 2 (Host, h)**.** *A device provides infrastructure for hosting services; a host is described as* $h = \langle l, R_h, M_h \rangle$*, where* $\underline{l}$ *represents the geospatial location of the device,* $\underline{R_h}$ *characterizes its entire processing resources, and* $\underline{M_h}$ *contains a list of metrics that gives evidence about ongoing operation.*

The current resource utilization is available as part of M_h. While hosts do not have an explicit representation of SLOs, their imperative requirement is to cap maximum utilization, e.g., maintain *cpu* load below 100%. The SLO fulfillment emerges from the deployments (e.g., hardware capabilities) and the current environment (e.g., service demand or network issues); given M_s, M_h, it is possible to evaluate all SLO.

The set of available hosts (H) is sampled from the CC's global pool of devices [20]; ideally, these hosts possess the desired characteristics (i.e., to fulfill SLOs), but depending on availability, it must optimize assignments for arbitrary hosts.

Definition 3 (Assignment, as)**.** *An assignment indicates that a service is executed on a specific device; the assignment is described by* $as = \langle s, h, R_s \rangle$*, where* $\underline{s}$ *is the service executed at host* $\underline{h}$*, and* $\underline{R_s}$ *is the share of the host's resources* (R_h) *utilized by executing* $\underline{s}$*. A short notation for any* as *is* $\underline{s} \diamond \underline{h}$*.*

Resources of multi-tenant devices are commonly partitioned into VMs and containers, i.e., services deployed at the same device do not access the same share of physical (processing) resources [18]. Nevertheless, the entirety of resources dedicated to n services assigned to a host h, plus all idle resources (R_I) equal the total amount of device resources (R_h).

$$R_h = R_I + \sum_{i=1}^{n} R_{s,i} \tag{2}$$

Each tenant is treated as an individual deployment; nevertheless, services interact indirectly by pooling resources from the same host. Hence, when assigning services over heterogeneous

devices, the question extends from "*which device can fulfill SLOs to the maximum degree*," to "*how much resources would individual services demand*." The precise implications of assigning a service to a host ($s \diamond h$) we call a "footprint"; both sides will be composed into one model, i.e., the MB.

B. Background

Bayesian Networks (BNs), as applied by Pearl [21], are structural causal models that can be represented as a Directed Acyclic Graph (DAG): edges between variables (e.g., $quality \rightarrow latency$) indicate conditional dependencies. For example, given that $quality = x$, what is the probability that $latency = y$, can in Bayesian terms be expressed as

$$P(lat \mid qual) = \frac{P(qual \mid lat) \times P(lat)}{P(qual)} \tag{3}$$

The causal edges in BNs are a distinctive feature that extends a system with explainability. Suppose we are interested in the behavior of a specific variable (x), we can explain x's state given its parents, children, and co-parent nodes. This subset is called its Markov Blanket (MB) – formally $\texttt{MB}(x)$ – and it disregards all variables that have no direct impact on x. Given its MB, a random variable is conditionally independent of all other variables, which is expressed by

$$P(x \mid \texttt{MB}(x), Y) = P(x \mid \texttt{MB}(x)) \tag{4}$$

Learning the structure of BNs and extracting MBs through data is not a simple task, many works are devoted to that; see [22] for a thorough survey on the topic. Regardless of their size, systems can be divided into smaller modules (i.e., MBs); thus, managed and controlled on a convenient scale. Within previous work, we built explainable MBs around SLO-governed components [7], [15]. To understand observable processes, we trained a causal representation of a system's states – all contained within a MB. Conditional variable dependencies could answer questions like "if *quality* rises, what is the probability that *latency* rises too?", expressed by Eq.(3). Given $\texttt{MB}(latency)$, conditional SLO fulfillment (i.e., $latency \leqslant x$) was analyzed to indicate the optimal system configuration (i.e., $quality = z$) that fulfills SLOs.

SLOs follow the grammar $var \rightarrow rel \rightarrow thresh$, where $var \in \{M_s \cup M_h\}$, $rel \in \{\leqslant, \geqslant\}$, and $thresh \in \mathbb{Q}$; hence, possible examples are $latency \leqslant 10$, or $cpu \leqslant 95$. The second SLO type is supplied as $obj \rightarrow var$, where $obj \in \{min, max\}$. These objectives represent soft boundaries that are optimized during operation, such as $min(energy)$. Recall, that SLOs characterize the QoS and QoE of individual services and their interfaces, i.e., the quality expected of in and out, but there is no knowledge how SLOs affect other services. By design, these MBs contain only internal system variables.

Extracting the MB around multiple variables forms larger subsets. For a service (s), its host (h), and m SLOs (Q), its MB is the composition of the subsets expressed as

$$\texttt{MBE}(M_s, M_h, Q) = \bigcup_{i=1}^{m} \texttt{MB}(\texttt{BNL}(M_s \cup M_h), Q_i) \tag{5}$$

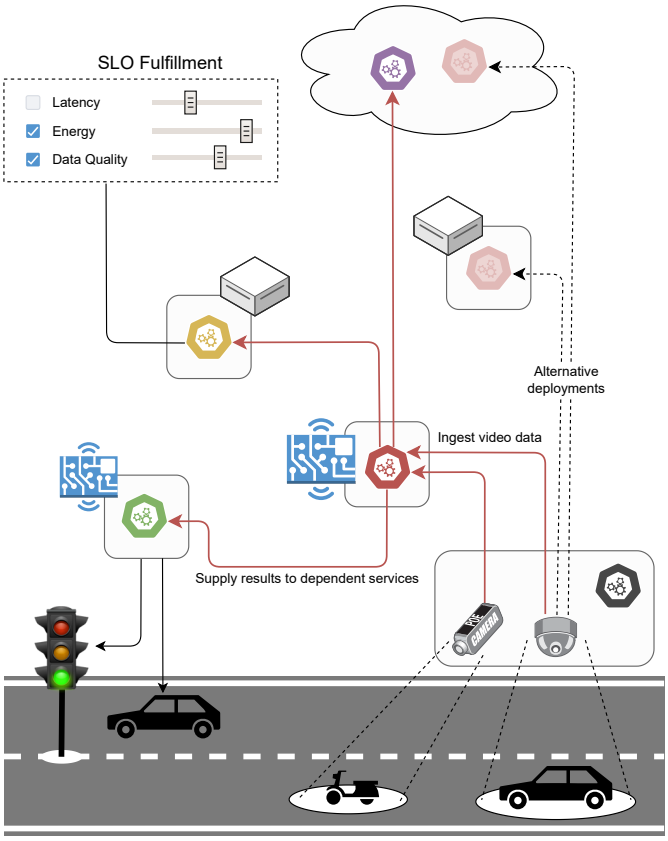


Fig. 2: Smart-city services chained together as a compound application

where MB and BNL are algorithms for MB extraction and BN learning, respectively, as presented in [15]; M_s and M_h contains multidimensional metrics monitored by executing $s \diamond h$. Whenever it is impractical to provide data for Bayesian Network Learning (BNL) upfront, or there occur variable shifts, our previous work [7] provides a remedy.

C. Illustrative Scenario

City spaces are increasingly crowded with sensor networks, most of them even publicly accessible [23]. This provides application developers access to vast amounts of data and allows them to combine and assemble smart city services at will. Figure 2 exemplifies how a service pipeline might be plugged together – data flows along the red errors. In that context, the following services and requirements (written in bold) could be envisioned by an application developer:

- *SmartCamera* (grey) provides batches of images from IoT cameras at a traffic junction. Can customize video quality.
- *RoadAnalysis* (red) analyzes video streams of a road scene. Can get congested with **high number of frames**.
- *TrafficRouting* (green) consumes information about traffic conditions and can control traffic lights or reroute vehicle navigation systems. Requires information **timely**.
- *TrafficPrediction* (yellow) assists local governments in estimating how traffic evolves throughout the day. Requires **large amounts** of **fresh** data to detect irregularities.
- *LiveMonitoring* (purple) allows remote inspection of analyzed road scenes. Requires **high video quality**.

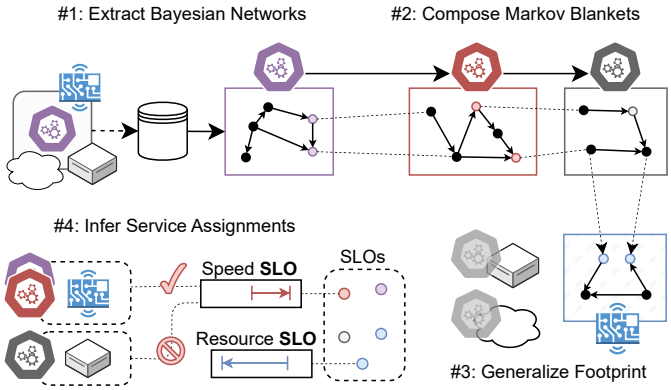


Fig. 3: Methodology for maximizing the SLO fulfillment of service pipelines by composing MBs and assigning services over heterogeneous CC hardware

The application developer combines the services as *SmartCamera* → *RoadAnalysis* → {*LiveMonitoring*, *TrafficPrediction*, *TrafficRouting*}; hence, the first two services in the pipeline provide their results to subsequent services. The immediate question here is how the requirements that each service poses constrain the QoS that *RoadAnalysis* and *SmartCamera* must provide – these are the transitive dependencies. The second question this unfolds is where to deploy individual services; preferably, application developers can stay agnostic in that regard and rely on the underlying CC infrastructure: depending on availabilities, services get assigned to hosts so that overall SLO fulfillment is maximized over the pipeline.

Both questions will be addressed as part of our methodology: application developers can customize *how* they would like their applications to operate (i.e., expressed in terms of SLOs) and rely on the assignment mechanism that finds the optimal hosting device for each service. To facilitate multi-tenancy for CC devices, this involves in-depth knowledge of the amount of resources required under a specific configuration.

III. Methodology

In the following, we present our 4-step methodology that assigns a microservice pipeline over heterogeneous resources, while maximizing SLO fulfillment. The sequential steps are embedded into the respective subsections III-A to III-D; to accompany explanations, Figure 3 provides a visual representation of the methodology: first, it (1) extracts MBs of individual services, then (2) performs MBC for the entire pipeline, (3) generalizes the service footprints, and (4) infers the assignment with maximum SLO fulfillment.

A. Markov Blanket Extraction

The training data for BNL is collected from past or ongoing operations; in the best case, services and their hosts were monitored over an extended period so that the underlying processes can be modeled accurately. Metrics of $s \diamond h$ are collected simultaneously (M_s, M_h); this tuple is appended to D, where the data of all empirically evaluated assignments is collected. To generate training data, each service should be executed at least once at an arbitrary host $h \in H$; to avoid perturbation through concurrent services, each service requires an isolated environment that gives clear evidence about how many resources (R_s) a service utilizes. Further, metrics of dependent services must be captured under equal configurations, e.g., if a pipeline contains two sequential microservices *CameraWrapper* → *StreetAnalysis*, the data produced by *CameraWrapper* must be the exact same received and processed by *StreetAnalysis*. This can be assured by (1) capturing both services' metrics at the same time, or (2) maintaining their interface variables aligned, e.g., by processing the same video resolution. If this is impractical or there is not sufficient data available, Section III-C can provide a remedy.

The BN training data is clearly created at the hosting devices; however, the infrastructure provider can choose freely where to execute any substep of the methodology, including the BNL. Ideally, it is a central location in the CC (i.e., low latency to all hosts) that can spare sufficient resources for training; otherwise, the methodology could be split up and executed separately by different hosts.

Given the training data (D), each tuple (M_s, M_h) is ingested to `MBE`, which trains the BNs and extracts the MBs around multiple SLOs (Q). This is also contained in Algo. 1, which serves as a wrapper for the 4-step methodology; nevertheless, Lines 2-5 are dedicated only to MB extraction. Afterward, X contains all empirically evaluated footprints. The remaining functions (i.e., `MBC`, etc) will be introduced in the next steps.

Algorithm 1 Wrapper for the 4 Methodology steps

Require: D, H, Q, $K = (S, E)$
Ensure: Z {Assignment with highest SLO fulfillment}
1: $X, V_d, V_c, F \leftarrow \varnothing$
2: {MB Extraction – Step 1}
3: **for each** (M_s, M_h) **in** D **do**
4: $\quad X \leftarrow \texttt{MBE}(M_s, M_h, Q_s) \cup X$ {Equals $s \diamond h$}
5: **end for**
6: $V_c, V_d \leftarrow \texttt{MBC}(E, Q)$ { – Step 2}
7: $F \leftarrow \texttt{FPG}(S, H, X)$ { – Step 3}
8: $Y \leftarrow \texttt{SASS}(E, V_c, V_d, H, Q, F)$ { – Step 4.1}
9: $Z \leftarrow \texttt{MAX\_AS}(Y, \text{“}joint\text{”})$ { – Step 4.2}
10: **return** Z

It is possible to logically separate the MBs of services and hosts, or rather, contained variables. However, none of the two exists without the other: service metrics (M_s) can only be observed during runtime; device metrics (M_h) give little insight when no service is executed. Hence, it makes no sense to train their MBs separately. Extracting the MB of a service also provides the impact on a hosting device – this is already their composition, what remains is to run MBC for services.

B. Markov Blanket Composition

Dependencies between services indicate that their internal states are influential to another service; thus, when aiming to maximize the SLO fulfillment of multiple services, it must be precisely determined *how* they affect each other. However, individual MBs cannot give evidence about variables external to their environment; also, training large BNs (e.g., [24], [25])

from one composite data set poses considerable requirements for BNL [7]. To this extent, we create an overarching system model by composing the MBs of pipelined services. This assembles a common behavioral model where it is evident how one service's state affects another service's SLOs.

We assume that changes to individual services perpetuate along the service pipeline – this path is encoded in the dependency graph (K). Given K, we test dependencies between any two services $\forall p, q \in S, (pq \in E)$; we compose the MBs of $\bar{E}$ instead of $\bar{S}^2$ intersections, i.e., only along the pipeline instead of pairwise for all $s_1, s_2 \in S$. For p, q and two variables $v_p \in M_p$ and $v_q \in M_q$, we test the statistical dependency between v_p and v_q. Therefore, we compute the difference between their probability distributions by applying two-sample Wasserstein distance (WSD) [26]. Given that there is a dependent variable v_d shared by p and q, Eq. (4) does not hold anymore; hence, their composition is linked by v_d.

Factors that, in reality, influence a service can be wrongfully missing from its MB because BNL could not consider them, e.g., the locations of dependent services are external. Such factors are also called confounding variables because they decrease model accuracy whenever they *would* have a conditional impact. Due to this, WSD is a good choice because it does not fail due to linear shifts in the distributions. For instance, $time(f_p(d))$ from Eq. (1) translates to $delay \in Q_q$, but $nl_{p \to q}$ cannot be represented without knowing both p's and q's location. Nevertheless, when testing their dependence, WSD can detect that they are dependent and that there is a linear confounder. Whether nl actually is *the* confounder is another question; however, the evaluation does not contain other confounders by design; hence, we will take it for granted in this paper and address this issue in future work.

More commonly, there is no confounder involved and two dependent variables can be directly mapped, e.g., a video $resolution \in M_p$ likely resembles the respective consumer's $v_q \in \texttt{MB}(q)$. To differentiate between confounded relations and the latter type, we compute the mismatch between states in the probability distributions. Since the variable distributions in the MBs represent discrete values, we use Jaccard similarity to quantify the divergence between states of v_p and v_q.

Pairwise dependency tests would have to cover $\texttt{MB}(p, Q_p) \times \texttt{MB}(q, Q_q)$ variable combinations; however, we are only interested in the implications to the successor's SLOs (Q_q) – reducing complexity to $\texttt{MB}(p, Q_p) \times Q_q$. For two dependent variables $v_p v_q$ with $v_q \in Q_q$, the SLO around v_q constrains the states that p can take without violating q's SLO; p does not differentiate between its "own" SLOs (i.e., Q_p) and transitive ones imposed by q – both restrict p's potential actions. Algo. 2 summarizes how the pipeline edges (E) are traversed to run pairwise dependency tests. The results are two lists of (simply) dependent (V_d) or confounded variable relations (V_c).

Algorithm 2 Markov Blanket Composition (MBC)

Require: E, Q
Ensure: V_c, V_d {Optimal assignment from all options}
1: **for each** (p, q) **in** E **do**
2: **for each** (v_p, v_q) **in** $mb(p) \times Q_q$ **do**
3: **if** $\texttt{WSD}(v_p, v_q) < 0.1$ **then**
4: **if** $\texttt{JD}(v_p, v_q) > 0.9$ **then**
5: $V_c \leftarrow v_q \cup V_c$
6: **else**
7: $V_d \leftarrow v_q \cup V_d$
8: **end if**
9: **end if**
10: **end for**
11: **end for**
12: **return** V_c, V_d

C. Generalize Service Footprint

The implications of running a service at a particular host are unique for this pair, i.e., how much and which resources are utilized. To find the assignment with the highest SLO fulfillment, a simple method is comparing the implications of all combinations [15]. Empirically testing all these pairs, however, is exhaustive for $S \times H$ combinations, but limiting potential assignments to those run empirically – X in Eq. (1) – must also be avoided. For example, $\{h_x, h_y\} = H$ and $\{s_x\} = S$ are available for assignment; however, there exists no empirical information on $s_x \diamond h_y$ – a likely situation if h_y was added recently to the device fleet – so that $s_x \diamond h_x$ is the only assessable assignment. The infrastructure provider, however, disposes of rich contextual information about services and devices, which allows estimating the SLO fulfillment of hypothetical assignments that were not tested empirically.

In particular, we use two information sources for this: The first is the metadata description of services [27], e.g., the primary purpose or the position in the pipeline. Our assumption is that services have similar hardware implications depending on their position (*pos*) in K, e.g., consumers located at the end have low hardware impact. The second source is a sampling mechanism for CC infrastructure [20], which provides a relative comparison of the hardware capabilities for devices in H; $h_1 > h_2$ implies that h_1 has more resources available. This information is used to extrapolate from empirically evaluated assignments (X) to hypothetical ones (F); hence, F will contain $(s \diamond h)$ for $\forall s \in S, h \in H$.

As depicted in Algo. 3, we distinguish four options to estimate the footprint $s_x \diamond h_x$ of any $s_x, h_x \in S \times H$: **(i)** the assignment was empirically evaluated (Line 3); **(ii)** a comparable service (i.e., same pos) was evaluated at h_x (Line 5); for this, we merge $s_x \diamond h\_$ (i.e., any host) and $s_y \diamond h_x$ by replacing all $\bar{M}_h$ variables of $h\_$ with those of h_x. If **(iii)** s_x was empirically evaluated at a weaker device (h_y), we reuse $s \diamond h_y$ because we assume it cannot perform worse on h_x (Line 3 **or**), whereas for $\forall h_y (h_y > h_x)$, we cannot know the implications to more constrained devices. This case can be adapted for comparable services as well – case *(ii)*. The last option **(iv)** is using the relative hardware difference (r) between h_x and an arbitrary host ($h\_$), where s_x was empirically evaluated, and penalizing the SLOs contained in

Algorithm 3 Footprint Generalization (FPG)

Require: S, H, X
Ensure: F {Generalized footprints for services $\diamond$ hosts}
1: **for each** s_x, h_x **in** $S \times H$ **do**
2: **if** $(s_x \diamond h_x) \in X$ **or** $\exists h_y (h_y \leqslant h_x \wedge s_x \diamond h_y \in X)$ **then**
3: $F \leftarrow (s_x \diamond h_x) \cup F$ {Case 1 and 3}
4: **else if** $\exists s_y (pos(s_y) = pos(s_x)$ **and** $s_y \diamond h_x \in X)$ **then**
5: $F \leftarrow \texttt{MERGE}(s_x \diamond h\_, s_y \diamond h_x) \cup F$
6: **else**
7: {assumes $\exists h\_ (s_x \diamond h\_ \in X)$}
8: $F \leftarrow \texttt{PENAL}(s_x \diamond h\_, r(h_x, h\_)) \cup F$
9: **end if**
10: **end for**
11: **return** F

Algorithm 4 Service Assignment (SASS)

Require: E, V_c, V_d, H, Q, F
Ensure: Y {Estimated SLO fulfillment per service $\times$ host}
1: **for each** $(s, \_), h$ **in** $E \times H$ **do**
2: $ext_s \leftarrow \forall v (v \in mb(s)$ **and** $v \in (V_c \cup V_d))$
3: **if** $(ext_s \cap V_c) \neq \varnothing$ **then**
4: **for each** $(s_1 h_1, ..., s_S h_H)$ **in** H^S **do**
5: $Y \leftarrow \texttt{INF}(F(s \diamond h), (Q_s \cup Q_h \cup ext_s \cup s_1 h_1)) \cup Y$
6: **end for**
7: **else**
8: $Y \leftarrow \texttt{INF}(F(s \diamond h), (Q_s \cup Q_h \cup ext_s)) \cup Y$
9: **end if**
10: **end for**
11: **return** Y

$s_x \diamond h\_$ according to r (Line 8). The interested reader finds implementations of MERGE and PENAL in the code artifact.

D. Service Assignment

The last step remaining is to identify the assignment that maximizes overall SLO fulfillment for all pipeline services; to that extent, we compare all hypothetical assignments and choose the one with the highest fulfillment as the sum

$$\sum_{s=1}^{S} Q_s + \sum_{h=1}^{H'} Q_h \tag{6}$$

where H' are all assigned hosts; notice that $|H'|$ can be 1 for a multi-tenant scenario. For $\forall s \in S$ and $\forall h \in H$, its footprint is retrieved by $F(s \diamond h)$ – this comprises the conditional variable assignments for the service's $\texttt{MB}(s, Q_s)$ and the respective hardware utilization of the host as $\texttt{MB}(h, Q_h)$. Transitive dependencies imposed by other services are contained in $\{V_c, V_d\}$; for a service s, all variables that share dependencies with other services are contained in ext_s (Line 2).

Given the footprints and the dependent variables, the SLO fulfillment of each service (i.e., constrained by ext_s) is obtained by ingesting the SLO thresholds (Q). As depicted in Algo. 4 (Line 5 & 8), we infer this from a footprint by providing variable assignments of the thresholds, and hypothetical deployments of dependent services. Internally, INF applies variable elimination[1]to marginalize all variables $\notin Q_s \cup Q_h$; hence, ending up with SLO fulfillment only. The distinction (Line 5 **or** 8) takes care of the confounded variable, i.e., $latency$. Estimating the conditional fulfillment of an SLO ($latency \leqslant x$) requires considering the hypothetical location of all services in the pipeline, i.e., H^S possible assignments.

Once SLO fulfillment under all possible assignments is inferred (Y), it remains to identify the optimal assignment; this is the role of MAX_AS. Here, our approach is to act either *greedy* or *joint*: *greedy* assigns services sequentially by marginalizing the hypothetical deployments of other services; hence, it estimates the expected SLO fulfillment of a service without considering where dependent services will be assigned, then, it chooses the host with the highest one. The *joint* approach, however, assigns services collectively by calculating the overall SLO fulfillment that all services would reach for a hypothetical assignment. Whenever a combination requires assigning multiple services to a single host h, the respective hardware utilization (e.g., CPU load) is appended to the existing one; as shown later in Table IV (red), such combinations can exceed the available resources, and hence, are disregarded because they violate the host's SLOs (Q_h).

When using *joint*, comparing all assignments in Y makes it evident which combination promises the highest SLO fulfillment. An advantage of *joint* is that it can precisely evaluate $latency$ SLOs and estimate the hardware utilization of multi-tenant assignments upfront, which avoids sub-optimal *greedy* assignments; however, the drawback of *joint* is its high combinatorial complexity. The interested reader will find the implementation of MAX_AS in the attached code artifact.

This concludes our 4-step methodology, which started by analyzing the dependencies between services and devices according to their MBs. To infer the optimal assignment between services and hosts, we use composed MBs and estimate the expected SLO fulfillment of different hypothetical assignments. In the next section, we now present how this methodology was implemented and evaluated.

IV. Evaluation

To evaluate the ideas presented in this paper, we focus on the individual steps of the methodology and highlight whether the outcome fulfills the research goals. For this, we first outline how the methodology and the evaluation environment were implemented; then we present the experimental setup (incl. services and hosts) and the results of our experiments. Lastly, we summarize and discuss the implications of these results.

A. Implementation

We provide a Python-based prototype[2]that implements our methodology; this includes all services used to generate BNL training data. To extract the MBs, our prototype requires tab-

[1]Variable elimination can infer knowledge from BNs. It repeatedly eliminates variables from a BN, while updating probabilities of remaining nodes, ending up only with a set of target variables – in our case the SLOs.

[2]Prototype artifact available at GitHub, accessed Feb 28th 2024

ular CSV files that are internally processed with pgmpy [28]; this step combines the data for each service, i.e., regardless of whether it was hosted at *Xavier* or *Orin*. This increases the general validity of trained BNs and adds conditional information on different device types, e.g., $P(cpu = x \mid type)$.

For $\forall p, q \in K; pq \in E$, we run pairwise variable tests to identify dependent variables; the required statistical tools (e.g., WSD) are native to Python. Whenever WSD $\leqslant 0.1$[3], we flag it as a potentially confounded relation. Depending on the Jaccard similarity, i.e., states match $\geqslant 90\%$, we declare whether it is confounded or simple. In both cases, additional constraints (i.e., external SLOs) are added to the dependent service's MB, which will be provided to `INF` (Algo. 4). Next, the footprint generalization is integrated into the inference: we iterate over all $S \times H$ combinations and estimate (1) expected SLO fulfillment and (2) hardware utilization. The default case is to perform *joint* service assignment, which internally compares assignments for H^S combinations; whenever this exceeds the maximum complexity, *greedy* is an alternative.

To determine the network latency (nl) between different hypothetical hosts, we rely on the knowledge of the infrastructure provider: we assume a tabular representation (i.e., $H \times H$) for this, which provides for $h_x, h_y \in H$ the respective $nl_{h_x \rightarrow h_y}$.

B. Experimental Setup

To embed and evaluate our implementation in a realistic setup, we rebuild the scenario presented in Section II-C; this means, that we use the five discussed services to instantiate our service set (S): *RoadAnalysis* was implemented and executed physically – for this we used the YOLOv8[4] model to detect and highlight objects within a road scene[5]. The other services were simulated based on the data ingested or produced by *RoadAnalysis*. According to services' position and purpose in the pipeline, we will use shorter synonyms: Producer (*P*) for *SmartCamera*, Worker (*W*) for *RoadAnalysis*, and Consumer ($C\_$) for {*LiveMonitoring*, *TrafficPrediction*, *TrafficRouting*}. Services depend on the data provided by their predecessor; hence, the dependency graph follows the inverse data flow from Figure 2, i.e., $K \sim \{C_1, C_2, C_3\} \rightarrow W \rightarrow P$.

SLO variables and thresholds are chosen according to the requirements in the service description. For simplicity, we assumed that $C_1, C_2,$ and C_3 have the same tractable variables; two of them are image $size$ and data $rate$, which reflect the video properties received by *Consumers*. Table I contains an overview of all SLOs: *W* and $C\_$ must ensure QoS SLOs, e.g., maintain $latency \leqslant x$ or ensure image $size \geqslant y$; *P* must minimize $energy$, i.e., a soft-boundary SLO, that is optimized as long as it does not violate any hard SLO (e.g., $latency$).

We provide a sampled set of devices (H) to host services, as shown in Table II: for each device, it contains a short ID, hardware stats, and how these stats (p, q) are classified relative to other devices in H. Given this information, it is evident how heterogeneous the devices are, e.g., the processing resources

[3]These two thresholds (i.e., 0.1 and 0.9) rendered satisfactory results.

[4]YOLOv8 model from ultralytics GitHub, accessed Feb 28th 2024

[5]Road racing video scene from YouTube, accessed Feb 1st 2024

TABLE I: SLOs inherent to each service

	P	*W*	C_1	C_2	C_3
latency	–	–	$\leqslant$ 1s	$\leqslant$ 70ms	$\leqslant$ 40ms
image size	–	–	$\geqslant$ 720p	–	–
data rate	–	–	–	$\geqslant$ 25f	–
delta	–	$\leqslant \frac{1}{fps}$	–	–	–
*energy**	$min()$	–	–	–	$min()$

of *Server* dwarf *Nano*'s. Additionally, devices equipped with a GPU can accelerate *Worker*'s video processing through NVIDIA Cuda; this underlines the importance that hosts have on services and SLO fulfillment. The last column contains the networking delay (nl): we assume that devices are perfectly aligned on a single line so that the nl between two hosts can be computed based on their nl to *Nano*, e.g., communicating from *Xavier* to *Orin* takes $5ms - 3ms = 2ms$.

We create different evaluation scenarios by repeatedly selecting subsets of $S \times H$, as visible in Table III: for each of the 8 scenarios, it shows available services and hosts. For example, for t_0, $\{P, W, C_1\}$ must be assigned over $\{S, L, O, X, N\}$. The scenarios could reflect different positions in time, where more or fewer hosts are available for different services. Nevertheless, we evaluate scenarios separately and do not update assignments at runtime. Depending on the service descriptions, we decide that P must always be assigned to *Nano* – the local device that bundled IoT video streams; further, we assume that smart-city infrastructure (i.e., fed by C_3) is located close to *N*, hence, C_3 must be assigned to any $h \in \{N, X, O\}$.

C. Results

We execute the experimental setup on the prototype of our methodology; first, we show the resulting MB composition, explain dependencies between services, and present the inferred assignments. Afterward, we assess the quality of the assignments according to three factors: (1) we observe their empirical SLO fulfillment at runtime, and (2) compare the runtime fulfillment to the expected fulfillment. Additionally, we provide (3) an exhaustive comparison of how inferred assignments score compared to all alternative assignments. These three factors evaluate our solution in terms of QoS and QoE and allow us to judge the feasibility of our approach.

1) Inference through MBC: Figure 4 shows the intermediary outcome after two steps, i.e., after MB extraction and composition. The purple, yellow, and green services represent the *Consumers*, red the *Worker*, and grey the *Provider*. The upper colored squares contain services' MBs, including SLO variables (fully-colored nodes) or such related to SLO fulfillment (black nodes). The blue squares contain the hosts' respective MBs, and how the services impact its variables. Dependent variables between services are represented by dashed lines and colored margins, e.g., $size$ (purple) is found dependent on $pixel$ (red), and in further consequence on $resolution$ (grey). Hence, constraining the service provided by *SmartCamera*. Now whenever red looks to infer a variable assignment that

[6]Prices adopted from sparkfun, accessed Feb 13th 2024

TABLE II: List of hosting devices used for implementing and evaluating the presented methodology

Full Device Name	ID	Price[6]	CPU	RAM	CUDA GPU	p [1,4]	g [0,3]	$nl_{N\to\_}$
Custom Server Build	*Server (S)*	2500 €	AMD Ryzen 7700 (8 core)	64 GB	RTX 3090	Very High (4)	High (3)	20 ms
ThinkPad X1 Gen 10	*Laptop (L)*	1700 €	Intel i7-1260P (16 core)	32 GB	————	High (3)	None (0)	10 ms
Nvidia Jetson Orin	*Orin (O)*	500 €	ARM Cortex A78 (6 core)	8 GB	Volta 383	Medium (2)	Medium (2)	5 ms
Nvidia Jetson Xavier	*Xavier (X)*	300 €	ARM Carmel v8.2 (6 core)	8 GB	Amp 1024	Medium (2)	Low (1)	3 ms
Nvidia Jetson Nano	*Nano (N)*	200 €	ARM Cortex A57 (4 core)	4 GB	————	Low (1)	None (0)	——

TABLE III: Services and hosts available for assignment

	Services					Hosts				
t_i	P	W	C_1	C_2	C_3	S	L	O	X	N
0	✓	✓	✓	–	–	✓	✓	✓	✓	✓
1	✓	✓	–	✓	–	✓	✓	✓	✓	✓
2	✓	✓	–	–	✓	✓	✓	✓	✓	✓
3	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
4	✓	✓	✓	✓	✓	–	–	✓	✓	✓
5	✓	✓	✓	✓	–	–	–	✓	–	✓
6	✓	✓	✓	✓	–	✓	–	–	–	✓
7	✓	✓	✓	✓	–	–	✓	✓	–	✓

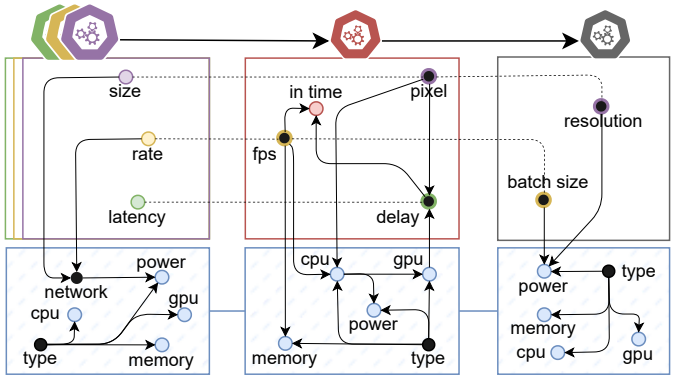


Fig. 4: Composed Markov blankets for *Consumers*, *Worker*, and *Provider*

fulfills its own SLO (i.e., in_time), it constrains this to states that fulfill purple's SLOs as well.

The "soft" SLO, i.e., $min(power)$, does not pose hard constraints to `INF`; however, given multiple assignments with equal SLO fulfillment, the one with lowest $power$ is chosen: For example, the MBC between P and any host has encoded that low $resolution$ and $batch$ decrease $power$; hence, from the parameter space that fulfills these SLOs, it chooses *480p* and *15fps*, i.e., the state with lowest $power$. Notice, the most influential variable for the host's MB is always the device $type$ – whether *Xavier* or *Server* hosts the service has a big impact on the conditional fulfillment of hosts' SLOs (Q_h).

We perform the remaining two steps of the methodology and provide the inferred assignments for each scenario in Table V. Each scenario's first line (i.e., *infer*) shows how services should be assigned (i.e., to optimize SLO fulfillment) given the available hosts. For example, at t_2, the pipeline $S = \{P, W, C_3\}$ had to be assigned over $H = \{S, L, O, X, N\}$; the inferred assignment is $\{P \diamond N, W \diamond O, C_3 \diamond X\}$, where $P \diamond N$ is preconditioned.

Table IV exemplifies why W was assigned to O at t_2: under the hood, the SLO fulfillment of all possible assignments (Y)

TABLE IV: Select assignment given service & host SLOs (t_2)

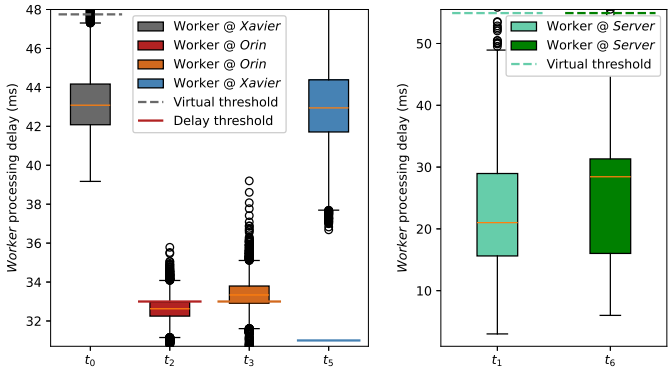
#	SLO Σ	W	CPU	GPU	Mem	C_3	Power Σ
1	1.70	*Orin*	50	30	119	*Orin*	8 W
2	1.52	*Orin*	24	30	73	*Xavier*	15 W
3	0.92	*Server*	3	31	12	*Laptop*	97 W
...	...	...	...	...	...	...	...
24	0.00	*Nano*	122	35	93	*Laptop*	26 W
25	0.00	*Laptop*	54	0	27	*Laptop*	21 W

TABLE V: SLO fulfillment of assignments (*infer* / *eval*)

		Services			
t_i	Mode	W	C_1	C_2	C_3
0	*infer*	$X: 0.99$	$O: 1.00$	–	–
	eval	$X: 1.00$	$O: 1.00$	–	–
1	*infer*	$S: 1.00$	–	$S: 1.00$	–
	eval	$S: 0.98$	–	$S: 0.99$	–
2	*infer*	$O: 0.70$	–	–	$X: 0.82$
	eval	$O: 0.75$	–	–	$X: 0.76$
3	*infer*	$O: 0.31$	$L: 1.00$	$L: 1.00$	$X: 0.36$
	eval	$O: 0.28$	$L: 1.00$	$L: 1.00$	$X: 0.28$
4	*infer*	$O: 0.32$	$X: 1.00$	$X: 1.00$	$O: 0.39$
	eval	$O: 0.29$	$X: 1.00$	$X: 1.00$	$O: 0.29$
5	*infer*	$X: 0.00$	$X: 1.00$	$N: 1.00$	–
	eval	$X: 0.02$	$X: 1.00$	$N: 0.99$	–
6	*infer*	$S: 1.00$	$S: 1.00$	$S: 1.00$	–
	eval	$S: 0.97$	$S: 1.00$	$S: 0.99$	–
7	*infer*	$O: 0.95$	$L: 1.00$	$L: 1.00$	–
	eval	$O: 0.99$	$L: 1.00$	$L: 0.99$	–
	Err ∅	0.02	0.00	0.01	0.06

was compared in a *joint* fashion. This evaluates the fulfillment given all 25 ($= H^2$) hypothetical deployments of W and C_3. In #1, we estimate that the collective SLO fulfillment (i.e., $Q_W + Q_{C3}$) is 1.7; however, this assumes that W and C_3 are both deployed at *Orin*, which is, on one hand, desirable because keeping W close to C_3 benefits its $latency$ SLO, but on the other hand, it is estimated that this would exceed *Orin*'s $memory$ (red cells). Hence, assignment #2 (green) promises the highest SLO fulfillment, whereas #24 shows that *Nano* would be incapable of running *Worker*, both in terms of service requirements and hardware limitations (orange).

2) Quality of Assignments: Apart from the expected SLO fulfillment, Table V also provides the experimental results of the assignment at runtime (*eval*). For each scenario, we tracked the services' performance at their respective hosts for 10 min, which generated in total roughly 70.000 observations. The last



(a) t_0, t_2, t_3, and t_5 (b) t_1 and t_6

Fig. 5: Processing delay of *Worker* at their assigned host $\in\{X, O, S\}$, combined with the threshold they must meet to fulfill all their consumers' SLOs

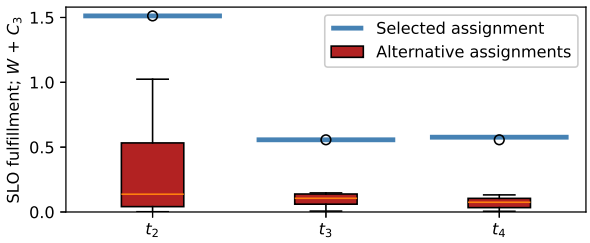


Fig. 6: Combined empirical SLO fulfillment of *Worker* and C_3 for the selected assignment, compared to the SLO fulfillment of all alternative assignments

table row contains the average prediction error (i.e., over all scenarios) between expected and actual SLO fulfillment.

Figure 5 shows the distribution of the processing $delay$ in different scenarios; the plots are separated due to the y-axis scale – *Server* has significantly lower processing $delay$ per frame. The solid lines express the threshold (i.e., minus overall nl) that W has to meet to satisfy the $latency$ of dependent *Consumers*. For example, in t_2 and t_3 the most restrictive $latency$ is imposed by C_3, i.e., $40ms$ in Table I; given the inferred assignment $\{W \diamond O, C_3 \diamond X\}$, we subtract $nl_{N\to O} = 5ms$ and $nl_{O\to X} = 2ms$, and set the bar to $33ms$. Table V confirms the validity of these distributions: t_2 reached 0.75 fulfillment and t_2 only 0.28. The difference between t_2 and t_3 occurs due to t_3 demanding higher $resolution$, which impacts $delay$ (see Figure 4). Dotted thresholds are virtual boundaries that fall outside the y-axis (i.e., 70 or $1000ms$).

Finally, Figure 6 shows the SLO fulfillment of the inferred assignment (blue line) in comparison to all alternative combinations. Since *P* does not pose any hard SLOs, we calculate overall SLO fulfillment as $Q_W + Q_{C3}$. The boxplots contain the 25 combinations of how these services can be deployed over H, which were all evaluated empirically over 10 min.

D. Discussion

This section summarizes presented results and highlights their implications: We report that (1) the MB extraction provided interpretable relations within individual MBs – the in_time SLO was correctly attributed to fps and $delay$ (see Figure 4); the MB composition was able to (2) detect dependencies between services and flag $latency$ as confounded due to the underlying nl. Further, we (3) correctly estimated that *Nano* exceeds SLOs from service and hosts (orange table cells), although W was not empirically evaluated at *Nano*.

Given the composed MBs, it could (4) consider the transitive SLOs imposed by other services – a good example is comparing t_2 and t_3 in Figure 5, which shows how $delay$ changed due to more restrictive SLOs from C_1. Further, we (5) maximized the SLO fulfillment given a heterogeneous list of hosts, e.g., t_1 in Table V could roam freely and save energy by assigning W to X, whereas for t_5 the best option was still unsatisfying due to tight constraints (from Table I). The fact that (6) we identified the optimal assignments (Figure 6) was mainly driven by low prediction errors (see Table V); ideally, this error will be fed back to improve predictions.

E. Limitations

While this paper presents a novel approach for raising SLO fulfillment of microservice pipelines, there remain limitations that must be addressed in future work; in the following, we will discuss three of them in more detail. Firstly, the initial process of determining the MBs of individual microservices can be computationally expensive and difficult to scale in large, dynamic networks. In order to avoid any overhead impeding regular device operation, it requires dedicated experiments that analyze the scalability of the approach. This must include a larger number of services and variables, as well as the methodology's performance on heterogeneous hardware.

Secondly, training an accurate BN and its corresponding MB requires substantial and high-quality data. In environments where data is sparse, noisy, or non-representative, the reliability of the MB and any respective inference decisions can be compromised. It remains to evaluate the presented approach in such an environment. Thirdly, IoT and Edge computing environments are often dynamic, with changes in node availability, service requirements, and network conditions. Static BNs might not adapt to such changes, making the MB outdated and less accurate over time. This issue was partly addressed in previous work [7], [29], which focused on capturing changes in variable distribution. Nevertheless, dynamic retraining of the BN structure remains an open challenge.

V. Related Work

In the context of this paper, we identified two main areas of related work that intersect with our research: (1) modeling large-scale BNs to estimate how system changes perpetuate or can be countered, and (2) optimizing service deployments for constrained devices according to QoS requirements.

A. Large-scale Bayesian Network Modelling

To assess the resilience of a pipeline system, Yazdi et al. [24] presented a dynamic BN that provides insights under which conditions QoS can be assured. Extending to compound systems, Chen et al. [17] provided a dynamic causality graph called *CauseInfer* that pinpoints issues during runtime. *CauseInfer* uses a two-tier mechanism that splits the system into

a device and a service layer. To trace fault propagation within a vehicle control network, Wang et al. [25] transformed a dynamic fault tree into a BN; this could infer the probability of faults under different hypothetical setups. Multiple works use BNs for anomaly detection in IoT systems [30]–[32]; however, they are more focused on detecting, instead of mitigating them. The largest BNs in comparable literature were provided by Mengshoel et al. [33] as a large-scale diagnostic system for simulating aircraft parts; however, individual blankets were separated without intersections. BNL is still an actively developing field, which is underlined by Kitson et al. [34]; they provide a comprehensive overview of BNL techniques and algorithms that help create accurate causal models.

Given these works, we conclude that BNs are primarily used for fault detection or system behavior prediction. Most graphs featured a single large model, which appears feasible given one central data set; however, in the CC, services (i.e., data sources) are distributed, and training large BNs poses high requirements to edge device. *CauseInfer* composes its model from multiple subgraphs; nevertheless, it lacks a representation of hardware utilization based on deployed services. Contrarily, our method assembles a model at the desired granularity.

B. QoS-Aware Deployment in the Computing Continuum

To find optimal services configurations for multi-tenant edge devices, Zhang et al. [10] presented *Octopus*, which predicts SLO fulfillment of two variables based on a deep neural network. To avoid resource contention, Qiu et al. [35] created *FIRM*, which predicts the resource usage of services for a multi-tenant device. Cardelli et al. [36] designed an autonomous elasticity mechanism for Cloud and Edge that ensures QoS of service chains; however, they did not implement it. Khoshkholghi et al. [37] presented a deployment and load-balancing mechanism that assured QoS of Edge function pipelines through deep learning. To maximize user satisfaction, Sheu et al. [13] propose a model deployment algorithm for the Edge that considers hardware limitations. The work of Zobaed et al. [12] allows to meet the latency constraints of multi-tenant applications. Confronted with the erratic activities of mobile users, Lu et al. [38] predicted how QoS could be assured through service updating. Their work provisioned services for multi-tenant deployments. To assure high QoS for composite services, Mehdi et al. [39] selected individual services based on a computed trust score. They would then construct a composite service through BNL.

Considering presented work, we conclude that multi-tenancy is common for the Edge; there are several works that estimate resource implications of services. However, none of them would consider the generalization of service footprints or transitive dependencies between services (i.e., tenants). The exception is [39]; however, they lack implications on the underlying hardware. Contrarily, our method provides the precise utilization per tenant for composite service pipelines.

VI. Conclusion & Future Work

This paper presented a statistical reasoning model for assigning a microservice pipeline over a heterogeneous set of devices, which are located from the Edge to the entire CC. To maximize the requirement fulfillment throughout a pipeline, our methodology analyzes dependencies between services; this constrains the operation of individual services according to the quality expected by dependent services. The evaluation of our prototype showed that we could infer the optimal assignments given a mutable list of services, hosting devices, and SLOs that had to be ensured. We envision our methodology as a central tool to simplify the development of compound services, e.g., in smart cities, where overall SLO fulfillment is optimized for whatever resources the CC has available. In that regard, future research will focus on runtime mechanisms that allow services to scale vertically or horizontally over the hosting devices.

Acknowledgment

Research received funding from the EU's Horizon Europe Research and Innovation Program under Grant Agreement No. 101070186. EU website for Teadal: https://www.teadal.eu/

References

[1] A. Rejeb, K. Rejeb, S. Simske, H. Treiblmaier, and S. Zailani, "The big picture on the internet of things and the smart city: a review of what we know and what we need to know," *Internet of Things*, Aug. 2022.
[2] N. Herbst, S. Kounev, and R. Reussner, "Elasticity in cloud computing: What it is, and what it is not," *International Conference on Autonomic Computing*, pp. 23–27, Jan. 2013.
[3] S. Dustdar, V. C. Pujol, and P. K. Donta, "On Distributed Computing Continuum Systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 4, pp. 4092–4105, Apr. 2023.
[4] M. Nardelli, G. Russo Russo, and V. Cardellini, "Compute Continuum: What Lies Ahead?" in *Euro-Par 2023: Parallel Processing Workshops*, 2024, pp. 5–17.
[5] P. K. Donta, I. Murturi, V. Casamayor Pujol, B. Sedlak, and S. Dustdar, "Exploring the Potential of Distributed Computing Continuum Systems," *Computers*, vol. 12, no. 10, p. 198, Oct. 2023.
[6] V. Casamayor-Pujol, A. Morichetta, I. Murturi, P. K. Donta, and S. Dustdar, "Fundamental Research Challenges for Distributed Computing Continuum Systems," *Information*, vol. 14, p. 198, Mar. 2023.
[7] B. Sedlak, V. C. Pujol, P. K. Donta, and S. Dustdar, "Equilibrium in the Computing Continuum through Active Inference," Nov. 2023.
[8] B. Sedlak, V. Casamayor Pujol, P. K. Donta, and S. Dustdar, "Controlling Data Gravity and Data Friction: From Metrics to Multidimensional Elasticity Strategies," in *IEEE SSE 2023*, Chicago, IL, USA, Jul. 2023.
[9] Y. Cao, "Better Orchestration for SLO-Oriented Cross-site Microservices in Multi-tenant Cloud/Edge Continuum," in *Proceedings of the 24th International Middleware Conference*, New York, USA, Dec. 2023.
[10] Z. Zhang, Y. Zhao, and J. Liu, "Octopus: SLO-Aware Progressive Inference Serving via Deep Reinforcement Learning in Multi-tenant Edge Cluster," in *Service-Oriented Computing*, Cham, 2023.
[11] Z. Xiang, S. Deng, J. Taheri, and A. Zomaya, "Dynamical Service Deployment and Replacement in Resource-Constrained Edges," *Mobile Networks and Applications*, vol. 25, no. 2, pp. 674–689, Apr. 2020.
[12] S. Zobaed, A. Mokhtari, J. P. Champati, M. Kourouma, and M. A. Salehi, "Edge-MultiAI: Multi-Tenancy of Latency-Sensitive Deep Learning Applications on Edge." IEEE Computer Society, Dec. 2022, pp. 11–20.
[13] J.-P. Sheu, Y.-C. Pu, R. Jagadeesha, and Y.-C. Chang, "An efficient module deployment algorithm in edge computing," in *IEEE Wireless Communications and Networking Workshops (WCNCW)*, Apr. 2018.
[14] J. Pearl, *Probabilistic reasoning in intelligent systems : networks of plausible inference*. San Mateo, Calif. : Morgan Kaufmann, 1988.
[15] B. Sedlak, V. C. Pujol, P. K. Donta, and S. Dustdar, "Designing Reconfigurable Intelligent Systems with Markov Blankets," in *Service-Oriented Computing*, 2023, pp. 42–50.

[16] V. C. Pujol, B. Sedlak, P. K. Donta, and S. Dustdar, "On Causality in Distributed Continuum Systems," *IEEE Internet Computing*, vol. 28, no. 2, pp. 57–64, Mar. 2024.
[17] P. Chen, Y. Qi, and D. Hou, "CauseInfer: Automated End-to-End Performance Diagnosis with Hierarchical Causality Graph in Cloud Environment," *IEEE Transactions on Services Computing*, 2019.
[18] D. Petcu, "Service Deployment Challenges in Cloud-to-Edge Continuum," *Scalable Computing: Practice and Experience*, Nov. 2021.
[19] V. Velepucha and P. Flores, "A Survey on Microservices Architecture: Principles, Patterns and Migration Challenges," *IEEE Access*, 2023.
[20] V. C. Pujol, A. Morichetta, and S. Nastic, "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)*, Jul. 2023.
[21] J. Pearl, "Causal inference in statistics: An overview," *Statistics Surveys*, vol. 3, no. none, pp. 96–146, Jan. 2009.
[22] M. J. Vowels, N. C. Camgoz, and R. Bowden, "D'ya like DAGs? A Survey on Structure Learning and Causal Discovery," Mar. 2021.
[23] M. Gascó-Hernandez, "Building a smart city: lessons from Barcelona," *Communications of the ACM*, vol. 61, no. 4, pp. 50–57, Mar. 2018.
[24] M. Yazdi, F. Khan, R. Abbassi, and N. Quddus, "Resilience assessment of a subsea pipeline using dynamic Bayesian network," *Journal of Pipeline Science and Engineering*, vol. 2, no. 2, p. 100053, Jun. 2022.
[25] C. Wang, L. Wang, H. Chen, Y. Yang, and Y. Li, "Fault Diagnosis of Train Network Control Management System Based on Dynamic Fault Tree and Bayesian Network," *IEEE Access*, vol. 9, pp. 2618–2632, 2021.
[26] A. Ramdas, N. Garcia, and M. Cuturi, "On Wasserstein Two Sample Testing and Related Families of Nonparametric Tests," Oct. 2015.
[27] A. Morichetta, V. C. Pujol, S. Nastic, S. Dustdar, D. Vij, Y. Xiong, and Z. Zhang, "PolarisProfiler: A novel metadata-based profiling approach for optimizing resource management in the edge-cloud continnum," in *18th Annual System of Systems Engineering Conference (SOSE)*, 2023.
[28] A. Ankan and J. Textor, "pgmpy: A Python Toolkit for Bayesian Networks," Apr. 2023.
[29] B. Sedlak, V. C. Pujol, P. K. Donta, and S. Dustdar, "Active Inference on the Edge: A Design Study," in *2024 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, Mar. 2024, pp. 550–555.
[30] M. Toğaçar, "Detecting attacks on IoT devices with probabilistic Bayesian neural networks and hunger games search optimization approaches," *Transactions on Telecommunications Technologies*, 2022.
[31] K. A. Shukla, S. Ahamad, G. Rao, A. J. Al-Asadi, A. Gupta, and M. Kumbhkar, "Artificial Intelligence Assisted IoT Data Intrusion Detection," in *ICCCT 2021*, Dec. 2021.
[32] M. Odiathevar, W. K. Seah, and M. Frean, "A Bayesian Approach To Distributed Anomaly Detection In Edge AI Networks," *IEEE Transactions on Parallel and Distributed Systems*, Dec. 2022.
[33] O. Mengshoel, S. Poll, and T. Kurtoglu, "Developing Large-Scale Bayesian Networks by Composition: Fault Diagnosis of Electrical Power Systems in Aircraft and Spacecraft," Jan. 2009.
[34] N. K. Kitson, A. C. Constantinou, Z. Guo, Y. Liu, and K. Chobtham, "A survey of Bayesian Network structure learning," *Artificial Intelligence Review*, vol. 56, no. 8, pp. 8721–8814, Aug. 2023.
[35] H. Qiu, S. S. Banerjee, S. Jha, Z. T. Kalbarczyk, and R. K. Iyer, "FIRM: An Intelligent Fine-grained Resource Management Framework for SLO-Oriented Microservices," 2020, pp. 805–825.
[36] V. Cardellini, T. Galinac Grbac, M. Nardelli, N. Tanković, and H.-L. Truong, "QoS-Based Elasticity for Service Chains in Distributed Edge Cloud Environments," in *Autonomous Control*, 2018.
[37] M. A. Khoshkholghi and T. Mahmoodi, "Edge intelligence for service function chain deployment in NFV-enabled networks," *Computer Networks*, vol. 219, p. 109451, Dec. 2022.
[38] S. Lu, J. Wu, P. Lu, N. Wang, H. Liu, and J. Fang, "QoS-Aware Online Service Provisioning and Updating in Cost-Efficient Multi-Tenant Mobile Edge Computing," *IEEE Services Computing*, 2023.
[39] M. Mehdi, N. Bouguila, and J. Bentahar, "A QoS-Based Trust Approach for Service Selection and Composition via Bayesian Networks," in *2013 IEEE 20th International Conference on Web Services*, Jun. 2013.