Proteus: Towards Intent-driven Automated Resource Management Framework for Edge Sensor Nodes

Shashikant Ilager, Daniel Balouek, Sidi Mohammed Kaddour, Ivona Brandic

Proteus: Towards Intent-driven Automated Resource Management Framework for Edge Sensor Nodes

Shashikant Ilager*, Daniel Balouek†, Sidi Mohammed Kaddour†, Ivona Brandic*
*Institute of Information Systems Engineering, TU Wien, Austria
†IMT Atlantique, Nantes Université, École Centrale Nantes
CNRS, Inria, LS2N, UMR 6004, F-44000 Nantes, France

ABSTRACT

Edge computing provides critical resources for various latency-sensitive applications, including, safety-critical monitoring systems that process large volumes of data from sensors and IoT devices, employing machine learning pipelines for effective and reliable analysis. Such applications are deployed on specially designed Edge Sensor Nodes (ESNs) that possess various sensors and limited computing power and support multiple data analysis tasks. ESNs encounter unique operational challenges, including intermittent power supplies, limited connectivity, and dynamic application and resource requirements, which complicate runtime management. Conventional resource management platforms like Kubernetes and KubeEdge are unsuitable for the dynamic needs of ESNs due to their reliance on centralized control and expected stable conditions. To bridge this gap, our paper introduces a data-driven resource management framework tailored for the autonomous adaptation of ESNs to diverse application and infrastructure requirements. We propose an intent-based mechanism that aligns application requirements, such as end-to-end latency, with infrastructure goals like utilization levels. This mechanism translates high-level intents into actionable low-level configurations, balancing the competing demands of various applications and resources, thereby guiding us toward a more robust and efficient application management system. We have implemented a prototype system, evaluated it on an experimental testbed, and demonstrated that our approach performs better than static-only optimization approaches with minimal impact on application performance.

KEYWORDS

Edge Computing, Internet of Things (IoT), Intent-driven Resource Management, Automated Application Management, Environmental Monitoring Applications

ACM Reference Format:
Shashikant Ilager*, Daniel Balouek†, Sidi Mohammed Kaddour†, Ivona Brandic*, *Institute of Information Systems Engineering, TU Wien, Austria, †IMT Atlantique, Nantes Université, École Centrale Nantes, CNRS, Inria, LS2N, UMR 6004, F-44000 Nantes, France, , . 2024. Proteus: Towards Intent-driven Automated Resource Management Framework for Edge Sensor Nodes. In *Workshop on AI and Scientific Computing at Scale using Flexible Computing Infrastructures (FlexScience '24), June 3–4, 2024, Pisa, Italy*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3659995.3660037

1 INTRODUCTION

Edge computing is a paradigm that brings the computing resources closer to the data sources and users, reducing latency and improving performance. It also enhances privacy by allowing on-device analytics and avoiding the transmission of sensitive data to remote clouds [8, 22]. Edge computing is enabling the development of smart applications in domains such as robotics, autonomous vehicles, Augmented Reality and Virtual Reality, smart assistants, chatbots, and more. Moreover, Edge infrastructures are increasingly used for safety-critical monitoring systems, such as surveillance, smart traffic systems, and environmental monitoring and sensing, by processing the data from sensors and Internet of Things (IoT) devices [18, 24]. Consequently, these monitoring systems use resource-intensive Machine Learning (ML) pipelines to pre-process the data and predict the targeted parameter. We call such devices as *Edge Sensor Nodes* (ESNs). Hence, using these Edge resources efficiently to support such safety-critical applications reliably is vital.

ESNs pose unique challenges for runtime management and operation. Most of these devices run on batteries or solar-powered energy sources, which lack reliable power supplies. Moreover, they are often connected with low bandwidth and limited network connections. They support multiple applications, processing large amounts of streaming data from multiple sensors. Furthermore, once these devices are installed, accessing them physically is often expensive or infeasible in many cases. Therefore, managing ESNs in a resource and energy-efficient, autonomous manner is necessary.

Current platforms and methods for managing Edge resources are inadequate for such deployments. For example, Kubernetes [2] and KubeEdge [1] are popular distributed operating systems or resource management systems in Edge and Cloud systems. These systems rely on centralized controllers and assume stable network and operating conditions. Likewise, some solutions have proposed lightweight IoT-Edge frameworks [10, 14, 24] that provide Cloud-like capabilities to Edge nodes. However, these solutions use rule-based static algorithms to manage the application workloads and resources at runtime, which are unsuitable for the dynamic environments that *ESNs* operate. This calls for an approach to managing ESNs that considers the requirements of multiple applications and deals with the complexities of ESNs.

This paper presents Proteus, a novel data-driven resource management framework for ESNs that can automatically handle multiple application and resource requirements. Our framework consists of two main components: (1) a method to monitor the dynamic application requirements and infrastructure goals and (2) an intent-based mechanism to fulfill both the application requirements and infrastructure goals. Our framework takes into account the cross-application effects on ESNs and balances the trade-offs between different requirements arising from multiple applications. In summary, the key contributions of this paper are:

- We **propose** an intent-driven framework for the automated management of ESNs and application requirements.
- We **develop** an algorithm for satisfying multiple high-level intents by taking runtime actions and configuring low-level resource and application parameters.
- We **implement** a small prototype on a real-world testbed and demonstrate the effectiveness of our approach compared to static baselines.

2 BACKGROUND, MOTIVATION AND RELATED WORK

2.1 Edge Sensor Nodes

Edge Sensor Nodes (ESNs) are specialized computing devices that integrate sensors, computing elements, and application processes to support real-time applications on the network Edge. These devices are different from conventional Wireless Sensor Networks (WSNs)-based systems [4] where data sensing, preprocessing, and analytics are performed at separate locations. WSNs may have two or three-layer architecture, where a sensor only senses and measures the elements and transfers the data to gateway nodes or remote Cloud nodes for processing [15]. In contrast, in ESNs, data sensing, preprocessing, and analytical tasks are performed on the same device. It connects to Cloud nodes only when needed to offload computational tasks or log events and results.

This architecture has emerged from the demand for real-time applications such as wildfire detection, air quality measurement, oceanic study, and many other scientific and real-world applications, where large amounts of data from multiple sensors have to be processed continuously in real-time to detect extreme and unpredictable events. Such applications can be broadly classified as Environmental Monitoring Applications (EMAs). Many of such systems, including Waggle sensor nodes [12] and WISP [3], are now deployed in multiple places to monitor environmental parameters.

2.2 ML and Environmental Monitoring Applications (EMAs)

Environmental Monitoring Applications (EMAs) mainly process and analyze large amount of data to identify patterns or predict events based on the monitoring data acquired from various sensors. EMAs that require batch processing use sophisticated physics-based models, and they often run in High Performance Computing (HPC) facilities, while real-time EMAs rely on Machine Learning (ML) models in their pipeline to process the streaming data and predict the outcome. Therefore, in this work, we focus real-time EMAs that use ML models.

A typical workflow of an EMA running in ESN comprises four primary tasks as shown in Figure 1. First, the Data Acquisition task reads the data from the corresponding sensors. The quality and frequency of the data sensing are configured according to the application requirements. Second, the Data Preprocessing task transforms the raw data into a standard format that is suitable for ML models. This task may include compressing the data and standardizing or normalizing the input parameters, as required by the ML models. Third is the prediction or data analytics task, which is the main task of the application. Here, ML models are used for application-specific parameter predictions or for other ML tasks such as classification or anomaly detection. Finally, the result of this analytics task is served to the stakeholders or application users. While not all EMAs may conform to this workflow, it provides a generic overview of a typical ML-driven workflow in EMAs deployed on the Edge.

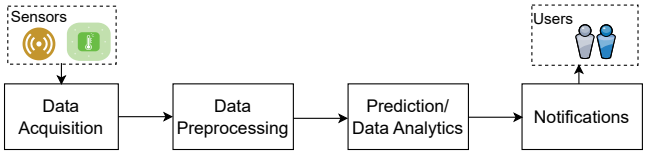

Figure 1: A typical workflow of ML-based monitoring applications in the Edge

2.3 Motivation

Managing *ESNs* deployed for safety-critical monitoring systems is extremely challenging due to multiple reasons. *First*, unlike Cloud data centers or Edge nodes inside micro data centers, where computing devices are deployed in controlled environments, ESNs are deployed in remote and outdoor locations. Such outdoor deployments are subject to limited (e.g., battery-powered devices) and interrupted power supply, unreliable networks, and other unfavorable environmental conditions [5, 27]. Thus, executing applications in a resource and power-efficient manner is crucial. *Second*, The ESNs are complex computing devices, which include computing elements and multiple sensors, such as cameras, environmental sensors, gyroscopes, and accelerometers, for sensing and measuring various environmental parameters. This introduces further complexities in managing resources and energy consumption. *Third*, these Edge nodes run multiple application workflows, where each application workflow processes streaming data from one or more onboard sensors, creating resource contention leading to poor performance and failures of workflow tasks. *Finally*, each application requires a different performance or Quality of Service (QoS), depending on the application, and real-world contexts perceived through onboard sensory systems. Therefore, executing such critical EMAs on resource-constrained complex Edge devices is a very difficult task.

2.4 Related Work

Resource Management in Existing Platforms: Existing solutions to manage Edge resources are mainly suitable for micro-data centers and Cloud data centers, which operate in controlled and reliable operating conditions. For instance, platforms such as Kubernetes [6] are used in controlled Edge and Cloud platforms where reliable deployment is expected. Kubernetes has many variants, such as KubeEdge [26], K3s, and MicroK8s [13]. These platforms extend Kubernetes and support distributed IoT application deployment of containerized applications. Some recent works also aim to provide automated resource management frameworks for IoT and Edge applications. R-pulsar [9, 21] is a framework that provides programming abstraction to deploy IoT-based applications on Edge by automating the decisions of where and when data needs to be processed. Tundo et al. [25] proposed a self-adaptive approach for energy-aware management of AI applications deployed on the edge

nodes using a state-based transition technique. Similarly, Oakestra [11] designs a hierarchical, lightweight, and scalable orchestration framework for Edge computing using multi-layer worker and manager components. These solutions expect a reliable centralized controller system to manage underlying resources, and they assume the presence of stable operating conditions in the Edge clusters and do not address the challenges of ESNs explicitly.

Intent-based Resource Management: Intent-based optimization has predominantly been applied within the networking domain, particularly in Software-Defined Networks (SDNs), commonly referred to as Intent-Based Networking (IBNs) [17]. Recent efforts have extended intent-based frameworks to the optimization of computing systems. For instance, Hui et al. [23] studied an intent-based cognitive continuum for resource optimization and adaptive management across the computing continuum. It provides a conceptual framework with various components for the intent-driven self-management of application workload and resources. Similarly, another conceptual framework and a proof of concept for serverless computing were studied in [20], demonstrating the integration of intent-based frameworks within the Edge-Cloud computing continuum. Metsch et al. [19] investigated Intent-Driven orchestration to comply with Service Level Objectives (SLOs) in Cloud deployments, integrating their solution with the Kubernetes platform and outlining a suite of tools—comprising a controller, planner, and actuators—that govern the life-cycle management of Kubernetes pods. Some studies have also explored joint optimization of network and compute resources in vehicular Edge computing platforms [16].

While these studies have explored diverse workloads, our focus is specifically on ESNs and Machine Learning (ML) pipelines within ESNs, which present distinct challenges. Additionally, the methodologies employed in the aforementioned studies are complementary to our approach. In the next section, we present our proposed approach and methodology in detail.

3 METHODOLOGY

This section defines the system model and describes architectural elements for our automated resource management approach.

3.1 System Model

Figure 2 presents a high-level overview of our proposed system architecture. Initially, applications are deployed within the Edge-Cloud Continuum, where each application defines its expected Quality of Service (QoS) and concurrently, the hardware infrastructure also defines its goals/objectives, representing its required operational state. Often, the requirements of applications and the goals of the infrastructure present conflicting trade-offs. Our system is designed to balance these trade-offs by simultaneously fulfilling application requirements and infrastructure goals through our middleware controller system. First, the ***infrastructure*** provides computational resources to deploy and execute application tasks on a distributed setup, where application workflow tasks are mainly executed on Edge nodes, and if required, Cloud nodes are used for our workflow tasks. Infrastructure requirements are defined as *Infrastructure Goals*. These goals could be available energy budget, execution cost, and average utilization, representing the desired state of operation mode. Consequently, *applications* consume data

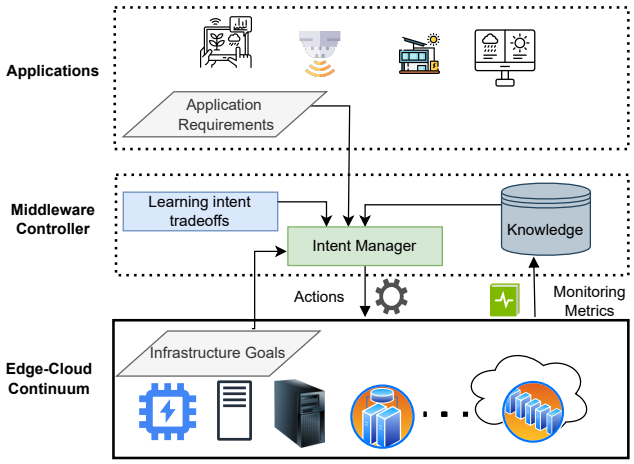

Figure 2: A high-level view of the proposed framework

from the sensors and execute workflow tasks. Here, we follow an ML-based application workflow as defined in Figure 2. Each application defines *Application Requirements* in terms of expected QoS. These requirements could be end-to-end latency, defined as the total time required to satisfy a user request, or the cost of computation or processing rate of user requests. Finally, both the application requirements and infrastructure goals are expressed as *Intents*, which are implemented with a ***controller*** system. A controller is a middleware system that configures application and infrastructure parameters, balancing both the *Application Requirements* and *Infrastructure Goals*. The main component of our controller is *Intent Manager*, which manages trade-offs between current intents based on system state.

In the following subsections, we describe the essential components of our system model in greater detail.

3.1.1 Infrastructure Model. We model our infrastructure as comprising a set of Edge Nodes (EN), where $E = \{E_1, E_2, ...E_m\}$ and Cloud Nodes (CN), where $C = \{C_1, C_2, ...C_n\}$. Thus, the aggregate resources available for application deployment are represented by $EN \cup CN$. Each Edge and Cloud node is distinct in terms of computational capabilities and available resources, offering a heterogeneous infrastructure for runtime deployment.

3.1.2 Application Model. The infrastructure facilitates multiple EMAs to process streaming data originating from sensors. We denote these applications by $A = \{A_1, A_2, ...A_u\}$, where each application comprises k microservices (as described in Section 2.2). Consequently, an individual application (A_i) is composed of microservices, $A_i = \{A_i^1, A_i^2, ...A_i^k\}$.

3.1.3 Intent Model. Intents are the high-level objectives or goals that application users and infrastructure providers would like to achieve. We define an Intent ($\mathcal{I}$), where $\mathcal{I} = \{ \mathcal{I}_1, \mathcal{I}_2, ... \mathcal{I}_v \}$, where $\mathcal{I}$ can be related to applications (and) or infrastructure. It is important to note that certain intents may inherently conflict with others, necessitating a balance of trade-offs. For instance, decreasing application latency could increase infrastructure resource utilization and energy consumption.

3.1.4 Actions. Actions are the mechanisms through which the system is configured. It requires a low-level resource and application parameters to align with the desired Intent at the time t. These intents are translated into actions by the controller. Specifically, in our framework, the controller undertakes the critical role of converting these high-level objectives into actionable low-level configurations.

3.2 Problem Formulation

The overall goal of our system is to satisfy a set of intents as $\mathcal{I}$, where each intent $i \in \mathcal{I}$ represents a specific objective or desired infrastructure state. As discussed earlier, intents may have dependencies or constraints that need to be satisfied. For example, if intent i_1 depends on intent i_2 being satisfied, we represent this constraint as: $i_1 \Rightarrow i_2$ Similarly, if intent i_3 requires both i_1 and i_2 to be satisfied, we represent it as: $i_3 \Rightarrow (i_1 \wedge i_2)$ The overall objective of the intent-based application is to satisfy the maximum number of intents while adhering to the defined constraints. We represent this as an optimization problem, where the objective function aims to maximize the number of satisfied intents.

$$\max \sum_{i \in \mathcal{I}} x_i \tag{1}$$

subject to the intent constraints, where x_i is a binary variable indicating whether intent i is satisfied (1) or not (0). However, solving this optimization function in Equation 1 is time-consuming and infeasible for real-time systems. Consequently, in the next section, we propose an online heuristic algorithm that finds an efficient solution in real-time.

3.3 Proteus: An Heuristic Approach for Intent-Driven Application Management

We demonstrate our method using a use-case scenario that involves three sample ML applications on a small-scale testbed, each with multiple intents and corresponding actions. Table 1 details all the crucial parameters and their values relevant to this study. Our infrastructure setup includes both Edge and Cloud nodes. The three ML-driven applications, which act as proxies for EMAs, comprise object detection, image classification, and image-to-text conversion. The three intents under consideration address application requirements (decrease latency, application priority) and infrastructure objectives (decrease resource utilization). As a result, we tailor our system to meet these intents through two primary actions: placement and processing rate adjustment. It is essential to note that while a broader range of intents and actions could be established based on the needs of the operational system, this study focuses on a select set of parameters to test and evaluate our approach.

A system can express its current intent; however, translating that into implementable actions necessitates configuring low-level application and resource parameters that influence the behavior of the system. This process includes two main steps: (1) Identifying the right configuration for the relevant intent requires domain-specific knowledge of the infrastructure and applications under consideration. (2) Resolving intent trade-offs, where the implementation of one intent could adversely affect other intents in place. For instance, decreasing resource utilization necessitates an action such as migrating a task bound to an ESN to a CN, which could impact the intent to decrease latency.

Translating Intent to Actions. We adopt a straightforward heuristic-based approach to translate high-level intents into actionable steps, as depicted in Algorithm 1. We adjust our system at each time interval t in our method. For every interval t, applications are sorted based on their current priority. Each intent follows the following actions. The application priority intent is given precedence over the other two intents, as priority indicates the urgency of an EMA based on real-world contextual settings. For the highest priority application, we designate the placement at the ESN, where all microservices of this application are exclusively located, and the processing rate is set to high. To decrease latency utilization intent, we maintain the placement of all applications at the ESN and set the processing rate to low. This strategy aims to provide a low-latency response to all applications, balancing it with the processing rate (number of requests per second). Conversely, for decreasing the resource utilization intent, all applications except the highest priority one have their placement configured to the CN and the processing rate set to default, indicating a lower operational utilization of the ESNs. While these actions are effective in ideal scenarios where an individual intent is applied at any given moment, the presence of multiple applications with varying intents necessitates resolving the constraints between intents.

Algorithm 1: Heuristic Algorithm for Intent Satisfaction

```
Input: I, Current state of system
Output: Set of actions, reorganized system
for t in time interval T do
    for i in Intent set I do
        A ←
          sortApplicationsinAscendingOrder(A.Priority);
        for A_j in A do
            configs ← IndentifyConfigurations();
            configs ← checkConstraints();
            applyAction(configs);
        end
    end
end
```

Resolving Intents Trade-offs. Since application priority takes precedence over the other two intents, we apply its actions straightforwardly, without modification. However, we need to adapt the actions of the other two intents (decreasing latency and resource utilization), which have direct trade-offs with each other. Actions related to decreasing latency and resource utilization that decide placement and processing rate configurations conflict with each other. To resolve this, we dynamically adapt our actions based on the current user workload and system state. We set the ESN's maximum and minimum CPU usage and processing rate threshold for each application. If an application's placement at the ESN is within the accepted processing rate threshold, we attempt to place the application's microservices at the ESN and follow the processing rate with the highest value among the two intents. In the case of a threshold violation, at interval t, we first halve the processing rate and observe the usage threshold since the placement decision

to CN drastically increases application latency. If halving the processing rate does not satisfy the threshold set, at the next interval t, we move the lowest priority application microservice to the CN, followed by others as necessary. Conversely, if the processing rate and CPU threshold are within limits, we try to increase the processing rate to the previous value and placement at the ESNs. This ensures that we continuously implement the intent to reflect the desired behavior for both application requirements and infrastructure goals, considering the dynamics of user workload level and resource usage.

4 PERFORMANCE ANALYSIS

4.1 Implementation

The proposed framework is implemented in the Python language, realizing Algorithm 1. Our controller thread creates a main thread that controls the business logic of the algorithm, while the three Edge applications are deployed as independent microservices and run inside dockerized containers. We decompose each application into two main microservices: data acquisition and inference. The preprocessing and notification microservices, as described in Figure 1, are part of the inference microservice since, in our setup, they require minimal processing cycles due to the use of curated existing datasets.

We employ the ImageNet dataset as a test dataset for all three applications. We generate workload for these applications using Locust [1]. The Edge node collects its own utilization metrics at predefined intervals using the "psutil" tool and updates the system state.

4.2 Experimental Setup

We have deployed our system on the Grid5000 cluster [7] , configuring it to emulate a real-world Edge Sensor Node (ESN) infrastructure. Specifically, we utilize two nodes: one serves as the ESN and the other as the Cloud Node (CN). To avoid expensive boot time of starting a container microservice, we run the same container images in both ESN and CN and dynamically route the requests. As user workload level varies, our controller dynamically routes requests between ESNs and CNs, representing a placement decision. Additionally, it adjusts the processing rate (number of concurrent users) in accordance with the controller's logic. We benchmarked three applications by running them in isolation on ESN to determine the maximum processing rate threshold. Table 2 displays the latency achieved by each individual application when the minimum processing rate (1) and the maximum processing rate (number of concurrent users, with each user rate set to 1) are selected, up until the point where the application microservices execute all requests without failures. Similarly, we set a minimum ESN's CPU usage threshold of 50% and a maximum of 90%. We set t to 5 minutes, and run the experiments for a total of 1 hour (T).

Baselines: We use the following two baselines to compare with our approach.

- **static_decrease_latency:** This is a static approach that aims to reduce the application end-to-end latency of all applications. It deploys all application services on ESN only.
- **static_decrease_resource_util:** This is a static approach that aims to reduce the overall resource utilization of ESN. It deploys application services belonging to the highest priority application on ESN and the other two applications' inference microservices on CN.

Workload Generation: For each application, the workload involves processing an image and providing inference results according to the specific ML model integrated within the application. In this process, each user request selects a random image from the test dataset, which is then sent to the application's inference microservices as a byte array via an HTTP POST request. The microservice preprocesses this payload, converting it into an input feature vector, which is subsequently passed to the deployed ML model for inference.

The volume of workload (measured in the number of concurrent users) for each application at time t is stochastically chosen within a range defined by a minimum of 1 and a maximum derived from the benchmarks presented in Table 2. This stochastic approach allows us to evaluate our framework under highly dynamic workload conditions.

4.3 Results and Analysis

We conducted the experiments three times and reported the average values in our results. We analyze the end-to-end latency, the failure rate of requests, and the average CPU utilization level of the Edge Sensor Node (ESN). Figure 3 illustrates the latency experienced by different applications across three approaches. The x-axis of the plot denotes the applications, while the y-axis quantifies the latency (in seconds). This latency is the average of all requests that received a response. The *static_decrease_latency* approach yields the lowest average latency across all applications, whereas *static_decrease_resource_util* exhibits higher average latency. This outcome is anticipated, as the ESN-only deployment avoids costly REST API calls with payloads to CN, thereby reducing total latency. Whereas our approach balances this trade-off, considering the current application context and priorities, and dynamically utilizes both ESN and CN for user requests, achieving, on average, better latency than *static_decrease_resource_util* with a marginal trade-off compared to the *static_decrease_latency* approach. However, the *static_decrease_latency* approach results in the highest request failure rate, as observed in Figure 4. This is attributed to ESN frequently experiencing resource overload conditions, causing the containerized microservices to fail in processing user requests, which then time out. Nevertheless, a significant failure rate is also observed with the *static_decrease_resource_util* approach, which, despite distributing requests more effectively than the static latency approach, still suffers from overloads at the CN, as two out of three applications receive requests during peak workload conditions at CN. On the other hand, by dynamically varying the processing of requests and routing them between ESN and CN at regular intervals, our approach experiences the lowest failure rate compared to the other two baselines.

Finally, we present the average CPU utilization of the ESN in Table 3. Effective resource utilization at the ESN is important, as ESN operates within limited computational power and energy availability constraints. As observed, the *static_decrease_latency* approach

[1] https://locust.io/

Table 1: The domain of the parameters used in our research problem.

Parameters	Definitions	Values
Infrastructure	Available resources including Edge and Cloud nodes	{Edge nodes (EN), and Cloud nodes(CN)}
Applications	List of applications in consideration	{Object-detection, Image-classification, Image-to-text}
Intents	High level objectives of applications and infrastructures	{Decrease Latency, Application Priority, Decrease Resource Utilization}
Actions	List of actionable measures to satisfy intents	{Placement, Processing rate }

App	Average latency (sec)	Max users (rate=1)	Max latency
Object-detection	0.38	110	1.75
Image-classification	0.25	210	1.35
Image-to-text	0.27	170	1.53

Table 2: Average end-to-end latency with minimum and maximum workload threshold to applications with Edge-only deployment when applications run in isolation on ESN.

	static_decrease_latency	static_decrease_resource_util	Proteus
CPU usage (%)	95.3%	73.1%	87.6%

Table 3: Average CPU usage of ESN during the experiments period

exhibits the highest average CPU usage at 95.3%, indicating near-full utilization consistently. In contrast, the *static_decrease_resource_util* approach demonstrates lower CPU usage at 73.1%, as it only processes the workloads of one application out of three in each time step t. Consequently, our approach achieves an average CPU utilization of 87.6%, utilizing the ESN effectively while avoiding resource overload, unlike the *static_decrease_latency* approach.

In summary, the results demonstrate that Proteus delivers improved Quality of Service (QoS) for applications by enabling dynamic adaptation of the system, taking into account runtime contexts, application requirements, and the state of the infrastructure.

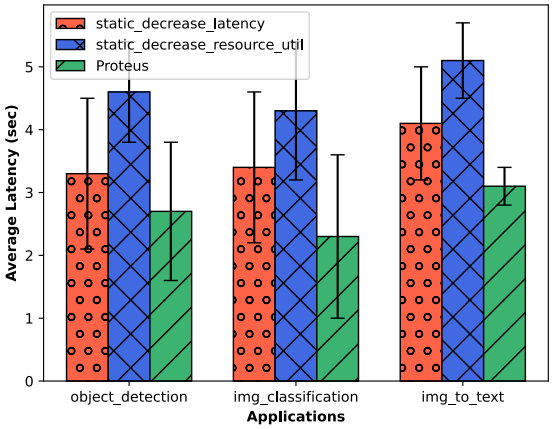

Figure 3: Average latency of different approaches during the experiments period. Monitored all user requests using the Locust workload generator.

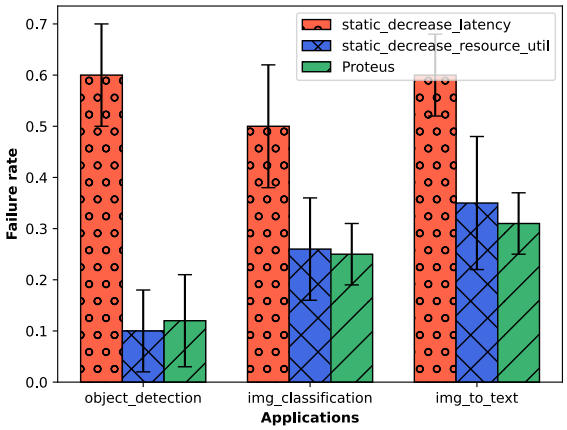

Figure 4: Average failure rate of user requests of different approaches during the experiments period.

5 FUTURE RESEARCH DIRECTIONS

The intent-driven approach represents a promising avenue for realizing an automated resource management framework within distributed systems. However, several challenges and limitations exist to achieve this goal successfully. Here, we will discuss the future works that tackle these limitations, aiming to seamlessly integrate the intent-driven mechanism into the management of distributed Edge and Cloud resources and applications at large.

5.1 A Standardized Framework to Define Intents

Intents are, by design, high-level abstractions meant to articulate the desired goals of system administrators and users. Currently, there are no standardized methods for defining an intent. Intents should be informed by the expected impact on relevant performance metrics and their interplay with other intents. We anticipate that existing methodologies like Resource Description Frameworks (RDFs) could facilitate the structured definition of intents and establish the ontology of a system and its dependencies. Nevertheless, existing RDFs are tailored for web resources and applications. Thus, broadening the RDF concept to include the definition of ESNs and CNs, their low-level resource parameters, and application workflows would aid in clearly defining and establishing structured relationships among multiple intents.

5.2 Compilation of High-level Intents into Low-level Configurations and Appropriate Actions

The high-level goals must be translated into low-level configurations across resources and application parameters in order to achieve an intent in runtime. Next, the values for these configurations must be estimated to guarantee that appropriate actions are carried out. For now, this translation is performed manually using heuristics directed by experts. However, an automated compiler is necessary for large-scale systems with thousands of heterogeneous nodes and application components in order to understand the intent and identify the set of possible resource and application parameters that must be adjusted while trading off with other intents and actions already in place.

This process would necessitate continuous monitoring of all parameters and systematic benchmarking of the system to collect a sufficient knowledge base. This would make it possible to build categorization or prediction models, which are essential for mapping the correct low-level configurations for a given intent. We also need optimization techniques that translate intents into actions. Metaheuristic or learning-based agent systems offer an interesting line of research to explore these issues.

5.3 Pluggable Interfaces for Integrating into Existing Middle-ware Tools

Contemporary resource management and middleware systems, such as Kubernetes and OpenStack, are designed to configure systems based on rule-based policies. However, intent-based resource management introduces new challenges, including the need to manage multiple application components for a single action and the requirement for continuous monitoring of both the system and the intents. These challenges necessitate the development of new software interfaces capable of monitoring and applying intent-driven actions in real-time. A component-based service composition will help to easily implement intent-driven middleware systems.

6 CONCLUSIONS

We proposed an approach for the automated resource management of Edge Sensor Nodes (ESNs) and managing ML-driven application workflows on them. We especially focused on Edge monitoring applications and addressing the challenges of managing them on ESNs. Our proposed framework, Proteus, considers the high-level objectives of application requirements and infrastructure goals and translates them into real-time actionable measures. This method offers an effective strategy for managing complex distributed ESN infrastructure while simultaneously meeting application demands. Our preliminary experiments and analysis of results indicate that our approach can reduce application latency and achieve better resource utilization compared to static-only methods. In future work, we aim to expand this work and develop a generic framework suitable for a diverse range of heterogeneous Edge-Cloud configurations and a wide set of Edge applications.

ACKNOWLEDGEMENTS

This work is partially funded through the projects *"Runtime Control in Multi Clouds" (Rucon)*, Austrian Science Fund (FWF): Y904-N31 START-Programm 2015; *"Transprecise Edge Computing" (Triton)*, Austrian Science Fund (FWF): P 36870-N; *Trustworthy and Sustainable Code Offloading (Themis)*, Austrian Science Fund (FWF): 10.55776/PAT1668223; *Sustainable Watershed Management Through IoT-Driven AI (Swain)* Austrian Science Fund (FWF): 10.55776/I5201, and by the Austrian Research Promotion Agency (FFG) through *Satellite-based Monitoring of Livestock in the Alpine Region (Virtual Shepherd)*, FFG Austrian Space Applications Programme ASAP 2022 #53079251. This work is partially funded through the French project OTPAAS (BPiFrance).

REFERENCES

[1] [n. d.]. KubeEdge. https://kubeedge.io/en/.
[2] [n. d.]. Kubernetes. https://kubernetes.io.
[3] [n. d.]. WISP: Wide-Area Infrared System for Persistent Surveillance. https://www.anduril.com/hardware/wisp/.
[4] Ian F Akyildiz and Mehmet Can Vuran. 2010. *Wireless sensor networks*. John Wiley & Sons.
[5] Atakan Aral and Ivona Brandić. 2021. Learning Spatiotemporal Failure Dependencies for Resilient Edge Computing Services. *IEEE Transactions on Parallel and Distributed Systems* 32, 7 (2021), 1578–1590. https://doi.org/10.1109/TPDS.2020.3046188
[6] The Kubernetes Authors. 2017. Kubernetes Documentation. https://kubernetes.io/docs/reference/ Copyright © 2017 The Linux Foundation®.
[7] Daniel Balouek, Alexandra Carpen Amarie, Ghislain Charrier, Frédéric Desprez, Emmanuel Jeannot, Emmanuel Jeanvoine, Adrien Lèbre, David Margery, Nicolas Niclausse, Lucas Nussbaum, Olivier Richard, Christian Pérez, Flavien Quesnel, Cyril Rohr, and Luc Sarzyniec. 2013. Adding Virtualization Capabilities to the Grid'5000 Testbed. In *Cloud Computing and Services Science*, Ivan I. Ivanov, Marten van Sinderen, Frank Leymann, and Tony Shan (Eds.). Communications in Computer and Information Science, Vol. 367. Springer International Publishing, 3–20. https://doi.org/10.1007/978-3-319-04519-1_1
[8] Daniel Balouek-Thomert, Eduard Gibert Renart, Ali Reza Zamani, Anthony Simonet, and Manish Parashar. 2019. Towards a computing continuum: Enabling edge-to-cloud integration for data-driven workflows. *International Journal of High Performance Computing Applications* 33 (11 2019), 1159–1174. Issue 6. https://doi.org/10.1177/1094342019877383/FORMAT/EPUB
[9] Daniel Balouek-Thomert, Ivan Rodero, and Manish Parashar. 2021. Evaluating policy-driven adaptation on the edge-to-cloud continuum. In *2021 IEEE/ACM HPC for Urgent Decision Making (UrgentHPC)*. IEEE, 11–20.
[10] Giovanni Bartolomeo, Mehdi Yosofie, Simon Bäurle, Oliver Haluszczynski, Nitinder Mohan, and Jörg Ott. 2023. Oakestra: A Lightweight Hierarchical Orchestration Framework for Edge Computing. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. USENIX Association, Boston, MA, 215–231. https://www.usenix.org/conference/atc23/presentation/bartolomeo
[11] Giovanni Bartolomeo, Mehdi Yosofie, Simon Bäurle, Oliver Haluszczynski, Nitinder Mohan, and Jörg Ott. 2023. Oakestra: A Lightweight Hierarchical Orchestration Framework for Edge Computing. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. USENIX Association, Boston, MA, 215–231. https://www.usenix.org/conference/atc23/presentation/bartolomeo
[12] Pete Beckman, Rajesh Sankaran, Charlie Catlett, Nicola Ferrier, Robert Jacob, and Michael Papka. 2016. Waggle: An open sensor platform for edge computing. In *2016 IEEE SENSORS*. IEEE, 1–3.
[13] Sebastian Böhm and Guido Wirtz. 2021. Profiling Lightweight Container Platforms: MicroK8s and K3s in Comparison to Kubernetes.. In *ZEUS*. 65–73.
[14] Laizhong Cui, Chong Xu, Shu Yang, Joshua Zhexue Huang, Jianqiang Li, Xizhao Wang, Zhong Ming, and Nan Lu. 2019. Joint Optimization of Energy Consumption and Latency in Mobile Edge Computing for Internet of Things. *IEEE Internet of Things Journal* 6, 3 (2019), 4791–4803. https://doi.org/10.1109/JIOT.2018.2869226
[15] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. 2013. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future generation computer systems* 29, 7 (2013), 1645–1660.
[16] TianZhang He, Adel N Toosi, Negin Akbari, Muhammed Tawfiqul Islam, and Muhammad Aamir Cheema. 2023. An intent-based framework for vehicular edge computing. In *2023 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. IEEE, 121–130.
[17] Aris Leivadeas and Matthias Falkner. 2023. A Survey on Intent-Based Networking. *IEEE Communications Surveys Tutorials* 25, 1 (2023), 625–655.

[18] Yi Liu, Chao Yang, Li Jiang, Shengli Xie, and Yan Zhang. 2019. Intelligent edge computing for IoT-based energy management in smart cities. *IEEE network* 33, 2 (2019), 111–117.

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

[20] Andrea Morichetta, Nikolaus Spring, Philipp Raith, and Schahram Dustdar. 2023. Intent-based Management for the Distributed Computing Continuum. In *2023 IEEE International Conference on Service-Oriented System Engineering (SOSE)*. IEEE, 239–249.

[21] Eduard Gibert Renart, Daniel Balouek-Thomert, and Manish Parashar. 2019. An edge-based framework for enabling data-driven pipelines for iot systems. In *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 885–894.

[22] Mahadev Satyanarayanan. 2017. The emergence of edge computing. *Computer* 50, 1 (2017), 30–39.

[23] Hui Song, Ahmet Soylu, and Dumitru Roman. 2022. Towards Cognitive Self-Management of IoT-Edge-Cloud Continuum based on User Intents. In *2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC)*. 1–4.

[24] Xiang Sun and Nirwan Ansari. 2016. EdgeIoT: Mobile edge computing for the Internet of Things. *IEEE Communications Magazine* 54, 12 (2016), 22–29.

[25] Alessandro Tundo, Marco Mobilio, Shashikant Ilager, Ivona Brandić, Ezio Bartocci, and Leonardo Mariani. 2023. An Energy-Aware Approach to Design Self-Adaptive AI-based Applications on the Edge. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 281–293.

[26] Ying Xiong, Yulin Sun, Li Xing, and Ying Huang. 2018. Extend cloud to edge with kubeedge. In *2018 IEEE/ACM Symposium On Edge Computing (SEC)*. IEEE, 373–377.

[27] Josip Zilic, Atakan Aral, and Ivona Brandic. 2019. EFPO: Energy Efficient and Failure Predictive Edge Offloading *(UCC'19)*. Association for Computing Machinery, New York, NY, USA, 165–175. https://doi.org/10.1145/3344341.3368818