SEA-LEAP: Self-adaptive and Locality-aware Edge Analytics Placement

Ivan Lujic, *Student Member, IEEE,* Vincenzo De Maio, *Member, IEEE,* Srikumar Venugopal and Ivona Brandic, *Member, IEEE*

Abstract—Near real-time edge analytics requires dealing with the rapidly growing amount of data, limited resources, and high failure probabilities of edge nodes. Therefore, data replication is of vital importance to meet SLOs such as service availability and failure resilience. Consequently, specific input datasets, requested by on-demand analytics (e.g., object detection), can be present at different locations over time. This can prevent exploitation of data locality and timely decision-making processes. State-of-the-art solutions for on-demand edge analytics placement either fail in providing low-latency access to user-requested input data or do not consider data locality. We propose SEA-LEAP (Self-adaptive and Locality-aware Edge Analytics Placement), a framework including a new mechanism for tracking data movements, on top of which we devise a generic control mechanism. SEA-LEAP enables on-the-fly placement of on-demand analytics considering the most appropriate dataset location that minimizes overall analytics requests execution time. We conduct experiments using real-world (i) object detection application, (ii) image datasets as input, (iii) self-designed benchmarks, and (iv) heterogeneous edge infrastructure using Kubernetes. Experimental results show the ability to efficiently deploy on-demand analytics and reduce total latency by 65.85% on average by performing adaptive data movements, indicating a promising solution for edge multi-cluster and hybrid environments.

Index Terms—Edge computing, analytics placement, data locality, distributed systems, autonomous, solution reference architectures.

1 Introduction

Modern IoT applications such as public safety video surveillance [1], predictive maintenance in smart manufacturing [2], and traffic management in smart cities [3], are characterized by strict latency requirements. Due to the rapidly increasing number of IoT sensing devices, data production is growing exponentially [4], with negative effects on the latency of analytics required by IoT applications. Edge computing, i.e., moving cloud processing closer to data sources, has been proposed as a solution to address these issues [5].

Still, the rapidly growing amount of data produced at the edge affects traditional centralized data collection and processing. Data can be transferred and replicated due to (i) limited storage capacities [6]; (ii) edge failure probabilities [7]; (iii) meeting certain service level objectives (e.g., data loss tolerance [8]); (iv) workload balancing [9]. Consequently, data can reside in locations different from where they were initially produced. Since exploiting data locality is crucial for latency-sensitive analytics requests, it is important to combine tracking of data movements and control logic for the timely placement of analytics applications.

Typical examples are edge video analytics applications. Performing video analytics (e.g., object detection to extract specific information of video frames) close to the source of data (e.g., on edge servers such as traffic cameras or micro data centers) is considered as the killer app for edge computing [10]. For example, video analytics on traffic footages of a specific area could be submitted to detect a suspect's vehicle. However, sampled footage frames from a traffic camera system can be stored at locations different from the source node to ensure fault tolerance, affecting the latency of on-demand analytics. This problem is present in other event-driven scenarios, where critical decision-making processes strongly depend on the timely placement of analytics requests, such as finding lost children or pets [1], and failure prevention in smart manufacturing [2]. Therefore, making self-adaptive analytics placement to the most suitable location is an important step toward improving the overall latency of decision-making processes.

In typical placement strategies for data processing applications available in the literature, researchers focus on traditional centralized data collection and analytics solutions [11], [12], or placing data processing based on resource-cost trade-offs [13], but do not discuss critical latency requirements of such analytics applications. Others propose strategies for latency-aware placement configurations of data stream processing applications [14] and low-latency data management on the distributed edge [15]. Many state-of-the-art approaches for analytics placement address the data locality from aspects such as edge-cloud workload balance [16], resource usage and query accuracy trade-off [17], or the fairness of cloud resource allocation [18]. Still, they do not consider the adaptive placement of on-demand analytics for low-latency access to user-requested data in the distributed edge environment. Data movement tracking and locality-awareness for scheduling on-demand analytics across distributed edge infrastructures are currently unsolved problems [19], [20].

To ensure efficient placement of on-demand analytics considering data locality and lower analytics execution time,

- *I. Lujic, V. De Maio and I. Brandic are with the Institute of Information Systems Engineering, Vienna University of Technology, A-1040 Vienna, Austria. E-mail: {ivan, vincenzo, ivona}@ec.tuwien.ac.at*
- *S. Venugopal is with the IBM Research Europe, Mulhuddart, Dublin 15, Ireland. E-mail: srikumarv@ie.ibm.com*

we propose SEA-LEAP, a framework for Self-adaptive and Locality-aware Edge Analytics Placement, featuring:

- **a new architecture**, enabling the exploitation of data locality based on the *tracking mechanism* that manages event-triggered registration of dataset movements across different edge nodes;
- **a generic control mechanism**, allowing on-the-fly adaptation and autonomous placement of on-demand analytics to node locations storing required input datasets.
- **a placement optimizer**, enabling on-demand analytics placement to the most appropriate dataset location that minimizes overall analytics requests execution time.

Overall execution time represents the completion time of the user analytics requests, which includes execution of tracking and control mechanisms to find the location that guarantees the lowest latency. We evaluate SEA-LEAP by conducting experiments on (i) real-world video analytics application; (ii) real-world sets of video frames as input for the application; (iii) obtained network and inference benchmarks, and (iv) heterogeneous edge infrastructure using Kubernetes platform. Experimental results show that SEA-LEAP is able to (i) autonomously deploy on-demand analytics requests (e.g., object detection) considering data locality, and (ii) reduce overall request execution time by 65.85% on average.

The main novelty lies in the combination of data movement tracking and self-adaptive control logic for analytics placement on different hardware, facilitating both code-to-data and data-to-code movements. Our paper advances the state-of-the-art approaches with a generic solution for deploying on-demand analytics while dealing with data locality issues. This can help users and developers to efficiently and timely deploy requested analytics across different edge infrastructures, indicating a promising solution for geo-distributed edge multi-cluster and hybrid environments.

We describe a motivational use case and the importance of data locality in Section 2. SEA-LEAP system design is proposed in Section 3, while tracking and control parts are detailed in Section 4 and Section 5, respectively. Section 6 shows the experimental setup, while Section 7 describes SEA-LEAP evaluation and discussion. Related work is outlined in Section 8. Section 9 concludes the paper.

2 Background

2.1 Motivational Use Case

We consider our InTraSafEd 5G project (Increasing Traffic Safety with Edge and 5G) [21], as the motivational example described in Fig. 1. It aims to improve traffic and pedestrian safety through video analytics running on edge nodes. Once data are collected, different analytics applications can query collected data. Example of these analytics ranges from (i) locating lost children or pets [1], (ii) timely locating suspects, (iii) finding a vehicle suspected of violating rules in captured footages (e.g., by recognizing license plates captured for accessing restricted central business districts, low emission zones), in a smart city. In such on-demand scenarios, critical decision-making processes strongly depend on the strict

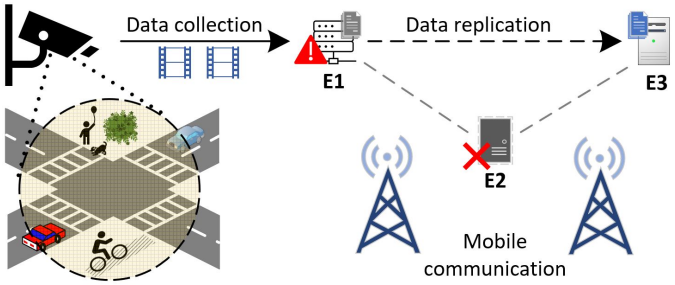


Fig. 1. An example smart city scenario to illustrate the problems of (i) tracking dataset movements/replications due to high edge failure probabilities, and (ii) timely placement of critical on-demand video analytics.

latency execution of edge analytics requests using specific input datasets.

However, edge servers can fail to execute analytics services for different reasons such as limited resources, power outages, or network failures [7]. Consequently, in the context of edge servers, it becomes necessary to replicate relevant data and services to other nodes (black dashed arrow), to meet Service Level Objectives (SLO), e.g., service and data availability. Based on calculated failure probabilities, some datasets can be replicated (partially or completely) to different locations to avoid data loss or interruption of running analytics services. To meet the low-latency requirements of on-demand analytics [22] (e.g., finding suspects), they should be placed at the same node where the dataset is stored to reduce the impact of network latency. However, in a geographically distributed edge infrastructure, replication causes the required dataset to be present in location(s) different from where it is produced.

Consequently, our challenges are to (i) keep track of datasets, and identify where they are located at a specific time point; (ii) identify which node location is the most suitable to reduce the latency of analytics requests. Inspired by the InTraSafEd 5G, we consider its benefits for running on-demand analytics on heterogeneous edge servers such as Raspberry Pi roadside devices with cameras attached to smart traffic lights.

Definition 1. *On-demand data analytics represents data processing applications that are submitted to a computational infrastructure $\mathcal{I}$ in response to specific user requests $\mathcal{R}$.*

The main characteristics of on-demand data analytics are: (i) they refer to specific input datasets [2], and (ii) they have low-latency requirements [22]. In this work, on-demand data analytics are represented as container-based applications running services that process input data and can be placed in different nodes of computational infrastructure (as in [23] and [24]). The input of on-demand analytics is a finite dataset that can be stored in different locations. We focus on a set of sampled video frames generated from video-camera systems.

2.2 Locality-aware Edge Analytics Placement

Edge computing is a paradigm where computation is performed on edge nodes, deployed near data sources. Edge computing is the key to exploit *data locality*, i.e., processing data closer to its origin, instead of collecting and processing

data far from its source [25]. Data locality is considered of paramount importance to meet low-latency requirements of on-demand analytics [26], [27]. However, identifying the correct dataset location to timely perform processing in the distributed edge is a challenging issue due to the possible data transfers and replications. Therefore, we introduce SEA-LEAP, a new framework allowing users and developers to easily deploy on-demand analytics applications without knowing the current location of the required datasets. Once required datasets are located by SEA-LEAP, on-demand analytics are automatically deployed to the most appropriate edge nodes, allowing analytics requests to meet low-latency requirements and significantly improving decision-making processes.

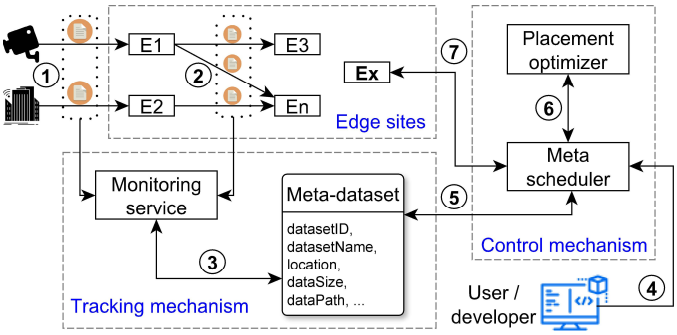


Fig. 2. SEA-LEAP Architecture Overview.

3 SEA-LEAP Design

The design concept of SEA-LEAP is driven by the following properties for on-demand edge analytics placement, namely,

- *Data locality-awareness*: as shown in the motivational scenario, data can change its locations over geographically distributed and heterogeneous edge nodes for different reasons, making it challenging to exploit data locality. Thus, the framework should be able to keep track of dynamic data movements and enable efficient locality-aware data management.
- *Autonomy*: on-demand analytics requests, as shown in the motivational scenario, often have low-latency requirements, making it difficult to timely identify the node minimizing overall request execution time. For this reason, the framework should be able to (i) find the most appropriate node location for analytics placement and (ii) handle numerous requests on time, with little or no human intervention in the deployment process. To this end, we need to enable autonomous analytics placements.
- *Genericity*: the computational edge infrastructure can be heterogeneous regarding both hardware resources and software configurations. Therefore, the framework design should be generic and applicable to work on top of existing systems by customizing the logic of proposed components and services, improving the overall reusability.

Fig. 2 provides an overview of the proposed SEA-LEAP architecture. We envision the on-demand analytics placement scenario based on data locality. To manage data locality-awareness with autonomous analytics placements, we illustrate three main parts:

Edge sites represent sets of geographically distributed edge nodes capable of executing on-demand analytics on data coming from IoT devices. IoT devices constantly generate data, which are transmitted to the edge infrastructure for temporary storage and future analytics.

Tracking mechanism is a component used for event-triggered registration of datasets and for tracking their future movements. It includes a monitoring service and meta-dataset that stores location-related details about datasets, enabling their dynamic tracking in the distributed edge. We focus on datasets with fixed sizes, which are generated, processed, and stored at the edge for future analytics. This is typical in storage-limited edge nodes, since in many cases it is enough to have a subset of data to preserve the analytics accuracy [28]. To minimize overall completion time, the control mechanism can trigger adaptive data movements.

Control mechanism is a component performing placement of on-demand analytics. It contains: the *meta scheduler* and the *placement optimizer*. The meta scheduler receives the description of on-demand analytics with the requested dataset name, and communicates with both the meta-dataset and the placement optimizer. The placement optimizer computes the most appropriate location for on-demand analytics to minimize overall execution time. Finally, the meta scheduler performs the actual placement to a target node and commands the analytics execution.

In Step 1, data generated from different IoT sensing devices are transferred to edge nodes, where they can be processed or stored for later analysis. In Step 2, datasets can be moved or replicated to other nodes due to different reasons such as fault tolerance. Any dataset generation, as well as its replication or movement are registered and updated constantly within the tracking mechanism (Step 3). For each dataset, current location-related details are stored in a database called *meta-dataset*. Meta-dataset can include information such as dataset id, dataset name, corresponding cluster, location path, and data size. Once a user submits the request description (Step 4), the meta-scheduler initiates the automatic placement adaptation. In Step 5, the meta scheduler extracts the required dataset name and retrieves location-related details of the required dataset from the database. We assume that a user knows the target dataset id or name needed as an input for requested analytics. Some of the proposed solutions include (i) access to a list of already generated and existing dataset names (e.g., based on a dataset catalog explained in Section 4), (ii) a consistent and regulated, easy-to-remember naming of datasets. Considering that (i) required dataset can be present in multiple nodes and (ii) different nodes can comply with analytics requirements (e.g., resource capabilities), in Step 6, the meta scheduler queries the placement optimizer to find the most appropriate location for analytics among node location candidates. Finally, the analytics application is placed to the most appropriate node (Step 7). The proposed SEA-LEAP follows the service-oriented architecture (SOA), featuring multiple parts and services that can be maintained independently. The following sections describe all parts in detail. Table 1 lists the main notations used in our approaches.

TABLE 1
Main Notations and Definitions

Notation	Description
α	An analytics application that requires input data.
d_{loc}	Variable representing the current dataset location.
d_{name}	Variable representing name of the dataset during the data generation at the data source
dma	A data management action (e.g., replication).
$meta_{db}$	Meta dataset database with location-related info.
KB	A knowledge base containing edge-relevant information.
rcv_{msg}	Variable containing request description for data mgmt.
rcv_f	Description containing the request for analytics placement.
L_{ap}	The most appropriate location for analytics placement.
$\mathcal{L}$	Matrix storing about node candidates for the placement.
$\mathcal{D}$	The set of datasets.
$\mathcal{N}$	The set of locations.
Λ	The set of nodes where dataset d is stored.
$l(n_i, n_j)$	Latency of network connection between n_i and n_j.
$b(n_i, n_j)$	Bandwidth of network connection between n_i and n_j.
$hops(n_i, n_j)$	Number of network hops between n_i and n_j.
$size(d)$	The overall size of a dataset d.
inf_time	An average inference time on a target node.
no_frames	Number of image frames in a target dataset d.

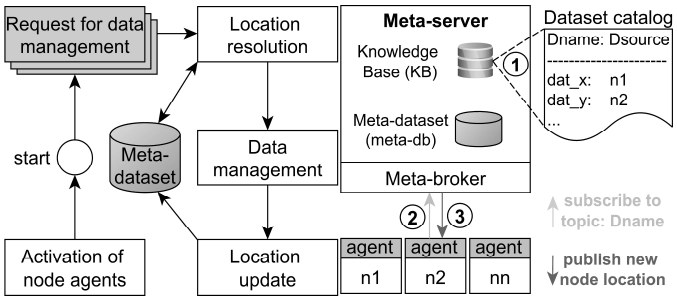
4 Tracking Mechanism

Based on the architectural model, we describe the tracking mechanism that is focused on data management and registration of edge data movements. Fig. 3 shows SEA-LEAP agent-based monitoring service, which incorporates an event-triggered registration of changes of data locations and publish/subscribe-based tracking of data movements, while Algorithm 1 shows pseudocode and the concept behind the agent-based monitoring. Regarding the *data locality-awareness* and *autonomy* properties, an autonomous software agent is employed on top of each node, constantly monitoring and acting upon data management events. Location-related details are stored in the meta-dataset database indicating where the datasets are currently available and accessible. The event-triggered data registration mechanism (see Fig. 3(a)) consists of several consecutive phases:

Activation of node agents. Every node has an agent listening to a known port (line 1, Algorithm 1). The $while$ loop (line 2) serves one client request for data management action. This phase is executed only once on each node and it is used in all the other phases. The received description of a data management action triggers the following phases.

Request for data management. In this phase, different requests for data management can be initiated (lines 3-4). We define data management as any data manipulation process including (i) generation of a new dataset, (ii) dataset replication or movement. Data management requests employ location-related details about data and can be initiated from (i) meta scheduler (Section 5.1), (ii) edge nodes, (iii) edge providers, or (iv) other incorporated mechanisms maintaining edge systems (e.g., load balancing, replication, fault tolerance).

Location resolution. In this phase, based on the target dataset, the location details are either (i) produced for newly generated datasets or (ii) checked in the meta-dataset before further actions. In the first case, a dataset is generated and stored in an edge node. Details about dataset location are saved in the meta-dataset and partially in the dataset catalog



(a) Event-triggered data registration (b) Pub/sub based data tracking

Fig. 3. SEA-LEAP monitoring service including (a) registration of changes of data locations and (b) tracking of data movements.

(see Fig. 3(b)). Dataset catalog (DC), a lightweight key-value store contains pairs of all generated dataset names and initial locations where they are created (Step 1). It supports the submission of the user's analytics request (see Section 5), and can be accessed from meta-server or keeping a synchronized copy locally. In the latter case (ii), the required dataset already exists and meta-dataset is queried, if needed, for retrieving location details.

Data management. In this phase, requested data management action (*dma*) is performed (line 5). Node agents complete data management requests. A trivial example $data_{mgmt}(fetch, dat\_x, n1)$ would fetch dataset dat_x from location $n1$. This phase is designed in a generic way, so it can also be adapted with other data management operations by edge infrastructure providers or deployed edge systems.

Location update. Once the previous data management actions are done, it is required to update new information in the meta-dataset. In this phase, a new connection to the meta database is established and corresponding dataset entries are updated (line 6). Once the database is updated, the source node of the corresponding dataset is notified via the pub/sub (publish/subscribe) channel enabling data movement tracking.

In case of failures, error messages will be returned in each phase. Furthermore, Fig. 3(b) shows the pub/sub based tracking of data management. First, as shown in the *location resolution* phase, DC stores existing and unique dataset names (Step 1) included in the Knowledge Base (KB). KB represents prior obtained and edge-relevant details such as edge-specific network topology, node characteristics, and analytics benchmarks. They provide a collection of information necessary for analytics placement (explained in Section 7). Edge node locations and produced datasets can be numerous and distributed. Thus, regarding the scalability of the tracking mechanism, the monitoring service includes a pub/sub messaging system in which every edge

Algorithm 1 `AgentBasedDatasetRegistration`

1: Listening for data management requests
2: **while** `TRUE` **do** ▷ serving one client request
3: Connect to client
4: $parse(rcv_{msg})$ ▷ analyse received message
5: $data_{mgmt}(dma, d_{name}, d_{loc})$ ▷ data management applied
6: Update $meta_{db}$, setting new node location d_{loc} for given d_{name}
7: Disconnect from client
8: **end while**

node ($n_1, n_2, ..., n_n$) can be subscribed to topics representing dataset names or ids that were initially produced at these nodes (Step 2). Once the dataset location is changed in the meta-dataset during the *location update* phase, the meta-broker publishes the change to a specific topic (Step 3). As a result, each node has information about the current location of its datasets, facilitating further edge data management actions. We assume a pub/sub system, such as MQTT (Message Queuing Telemetry Transport), due to its communication scalability and minimum resource requirements [29]. Note that the tracking mechanism is essential for enabling data locality-awareness, while the following control mechanism primarily ensures the self-adaptive and timely placement of edge analytics based on data locality. The scalability of the tracking mechanism will be investigated in future work.

5 Control Mechanism

The control mechanism is the cornerstone of SEA-LEAP architecture. The goal is to enable self-adaptive placement of an analytics application α considering a dataset location d_{loc}, based on two actions, namely,

- *GuideMe:* static placement of an analytics application to the source node candidate initially storing the requested dataset. If the input dataset is simultaneously present on multiple locations, the node ensuring the lowest estimated analytics execution time is selected as the target one;
- *FollowMe:* dynamic placement of an analytics application to an alternative node candidate that minimizes overall request execution time. In this case, an adaptive dataset movement from the source to the alternative node is necessary, before the analytics placement and execution.

These actions can offer a continuous adaptation of analytics placements in highly distributed and networked edge servers. Both actions rely on two important services of the control mechanism, namely, the *meta-scheduler* and the *placement optimizer*, described in the following.

5.1 Meta Scheduler

Accessing the dataset locations can be done by storing location details in the meta-dataset (described in Section 4), while placement adaptations are managed on-the-fly within the meta scheduler. We assume that the meta scheduler is accessible, and there can be multiple instances serving. Algorithm 2 describes the meta scheduler life cycle in detail. The scheduler continuously listens for new requests in lines 1-2. Once a user sends an analytics request description, containing details for application execution, its format is checked. If its format is valid (lines 3-5), the meta scheduler extracts information such as the name of the required dataset id and analytics process α (line 6). Next, the meta-dataset is checked and relevant information is retrieved (lines 7-8). If corresponding information exists, the meta scheduler will send details to the placement optimizer (line 9). The output of the placement optimizer represents the node location that guarantees the lowest total latency for the request and it is stored in the L_{ap} (lines 10-11). In case

Algorithm 2 `MetaScheduler`

```
1: while TRUE do                                   ▷ loop serving one client's requests
2:     rcv_f ← waitConnection()                    ▷ waiting for incoming connections
3:     if rcv_f = fmt then                         ▷ checking the format of analytics request
4:         continue
5:     end if
6:     parse(rcv_f)                                ▷ extracting needed information (d_name, α)
7:     Create matrix Λ with location-related information about d
8:     Λ ← retrieve(d_name)                        ▷ retrieving details from meta_db
9:     if Λ ≠ empty then
10:        L_ap ← PlacementOptimization(Λ)
11:        Adapt deployment templates with the L_ap and other info
12:        if L_ap is not one of the initial location from Λ then
13:            Replicate dataset d to L_ap using Algorithm 1
14:        end if
15:        deploy(R, L_ap)                         ▷ placing analytics to node location L_ap
16:    end if
17: end while
```

the usage of L_{ap} requires adaptive data movement, the meta scheduler will follow the procedure from Algorithm 1 (lines 12-14). Finally, the meta scheduler performs the placement of α in the node L_{ap} (line 15).

5.2 Placement Optimizer

The goal of the placement optimizer is to find an edge location that minimizes the total latency. It is designed to satisfy users' latency requirements for timely decision-making processes. Total latency is impacted by (i) analytics execution time that depends on node and dataset characteristics, (ii) data transfer that is affected by network characteristics.

We consider analytics placement as a minimization problem with latency as an objective. We assume that users can submit requests for executing data analytics applications over different input datasets, whose location is unknown to the user. Computational infrastructure is modeled as a graph $\mathcal{I} = (\mathcal{N}, \mathcal{E})$ (as used in [30]), such that $\mathcal{N}$ is a set of different nodes where applications and datasets can be placed, and $\mathcal{E}$ models the network connections between nodes. For each $(n_i, n_j) \in \mathcal{E}$, with $n_i, n_j \in \mathcal{N}$, we define both latency $l(n_i, n_j)$ and bandwidth $b(n_i, n_j)$ measurements of network connection (Section 6.3).

We also define a set $\mathcal{D}$ of different generated datasets, that can be initially stored in one or multiple nodes $n \in \mathcal{N}$ over a geographical area. In the latter case, we assume that those datasets are always synchronized. Further, each dataset d is defined by its size $size(d)$ and the set $\Lambda(d)$ of nodes where d is stored. Users submit a request $\mathcal{R} = (\alpha, d_{name})$ to $\mathcal{I}$, where α is an analytics application, and d_{name} is the name of the input dataset for α. For each $\mathcal{R}$, SEA-LEAP goal is to identify location L_{ap} for α and d that minimizes $\mathcal{R}$ total latency, i.e.,

$$L_{ap} = \arg\min_{\mathcal{L}[i]} (\mathcal{L}[i]^{t\_lat}_{\vec{R}}), \;\; L(\alpha) = L(d). \tag{1}$$

$\mathcal{L}$ is a matrix that contains location candidates with calculated total latency, including a potential data transfer and the analytics execution time on the specific node candidate:

$$TL = t^{mv_d}_{(n_i, n_j)} + t(R, n_j), \tag{2}$$

where n_i initially stores the dataset d ($n_i \in \Lambda(d)$), and n_j is an alternative node candidate ($n_j \in \mathcal{L}(d)$). In case of n_i as the initial candidate, i.e., $n_i = n_j$, then $TL = t(R, n_i)$.

TABLE 2
Edge Node Types Used in the Experimental Setup, Technical Details and Inference Latency Benchmarks for Each Node Type.

Node label	Node type	CPU	RAM	# of nodes	Edge TPU	Inference/frame [ms]
A	Raspberry Pi 4	Quad-core Cortex-A72 (ARMv7) at 1.5GHz	4GB	2	yes	17.81
B	Raspberry Pi 4	Quad-core Cortex-A72 (ARMv7) at 1.5GHz	4GB	1	no	250.74
C	Raspberry Pi 3 B+	Quad-core Cortex-A53 (ARMv7) at 1.4GHz	1GB	8	no	500.62

Otherwise, we define $t^{mv_d}_{(n_i,n_j)}$ as the time required to send d from n_i to n_j, i.e.,

$$t^{mv_d}_{(n_i,n_j)} = l(n_i,n_j) + hops(n_i,n_j) \cdot \frac{size(d)}{b(n_i,n_j)}, \quad (3)$$

where $hops(n_i,n_j)$ is the number of hops between n_i and n_j. Further, we define $t(R,n_j)$ as the estimated time required to complete analytic request $R(\alpha,d)$ on node n_j, i.e.,

$$t(R,n_j) = inf\_time(n_j) \cdot no\_frames(d), \quad (4)$$

where $inf\_time(n_j)$ is an average inference time per frame, and $no\_frames(d)$ is the corresponding number of frames in target dataset d (see Section 6.3).

Algorithm 3 describes the placement optimizer. First, two data structures are created to store location candidates for analytics placement (lines 1-2): $\mathcal{L}$, containing a total estimated latency, and $\mathcal{L}_{KB}$, which stores potential candidates retrieved from KB if they satisfy certain conditions, e.g., nodes with better inference time compared to the source node(s) (line 3). Next, each node containing the requested dataset (line 4) will initially become a candidate for placing the required analytics (lines 5-7). Then, even if the required dataset is present on multiple nodes, the placement depends on the total latency of each candidate (line 6). In lines 8-12, we calculate total latency for each new candidate, since due to the heterogeneity of the infrastructure it is possible to achieve a lower total latency on a node with more resources, despite the data transfer (line 10). Consequently, the most appropriate node location L_{ap} is the one offering the lowest total latency (line 14), returned in line 15. We consider three scenarios: (i) the dataset is stored on a single node with the lowest estimated analytics latency; (ii) the dataset is stored on multiple and heterogeneous nodes, therefore the most powerful will run the analytics; and (iii) edge node(s) storing the required dataset do not have resource capabilities to meet latency requirements, thus, the dataset will be placed to a node which minimizes overall request execution time.

Algorithm 3 `PlacementOptimizer`

Input: Set of nodes storing req. dataset Λ
Output: The most appropriate location L_{ap}
1: Create matrix $\mathcal{L}$ forming location candidates with est. total latency
2: Create matrix $\mathcal{L}_{\mathcal{KB}}$ for potential location candidates from KB
3: $\mathcal{L}_{\mathcal{KB}} \leftarrow KB(n_{type} > \Lambda(n_{type}))$ ▷ retrieving alternative candidates
4: **for** each $n_{init} \in \Lambda$ **do**
5: Add node n_{init} to $\mathcal{L}$
6: Calculate $TL(n_{init})$ for the initial node using (4)
7: $\mathcal{L}(n_{init}, TL) \leftarrow TL(n_{init})$
8: **for** each $n_{new} \in \mathcal{L}_{\mathcal{KB}}$ **do**
9: Add node n_{new} to $\mathcal{L}$
10: Calculate $TL(n_{new})$ for new nodes locations using (2), (3) and (4)
11: $\mathcal{L}(n_{new}, TL) \leftarrow TL(n_{new})$
12: **end for**
13: **end for**
14: $L_{ap} \leftarrow \mathcal{L}_i$ with minimum total latency ▷ Compute L_{ap} using (1)
15: Return L_{ap}

Theorem 1. *SEA-LEAP complexity is $O(n \cdot m + k)$.*

Proof. SEA-LEAP complexity is determined mainly by `MetaScheduler` and `PlacementOptimizer`. `MetaScheduler` complexity (see Algorithm 2) depends on `PlacementOptimizer` (see Algorithm 3), since all other lines have complexity $O(1)$. The for loop (line 4) from Algorithm 3 iterates over the set of node locations Λ that simultaneously store the requested dataset. Entering the inner for loop in line 8, it iterates over each new node location candidate and calculates the estimated total latency in line 10, resulting in complexity of $O(n \cdot m)$, where n is the number of replicas, representing the initial node candidates, and m is the number of alternative node candidates. Next, searching the candidate with the lowest total latency in line 14 has the complexity of $O(k)$, where k is the total number of node candidates. Other lines are $O(1)$, resulting in the overall complexity of $O(n \cdot m + k)$. □

$O(n \cdot m + k)$ is acceptable in our context, since (i) n is expected to be either 1 due to limited edge storage capacities [6] or small while still guaranteeing the resilience with fewer replicas as showed in [7], and (ii) m is limited to both nodes whose types match the types from KB benchmarks and that have lower inference time per frame than initial nodes from the set Λ.

6 Experimental Setup

SEA-LEAP is implemented using Python, while the experimental evaluation of the proposed SEA-LEAP architecture uses Kubernetes for deploying analytics applications. Our emulation-based evaluation is based on real traces and using RuconLiveLab, our physical edge infrastructure consisting of 11 Raspberry Pi (RPi) single-board computers, available in three different configurations (see Table 2).

6.1 Implementation Details

We first introduce technologies used for the experimental evaluation of SEA-LEAP. Considering the deployment of analytics applications, many researchers and industries are revealing nowadays the rapid adoption of Kubernetes orchestration platform [23], [24], relying on master-worker architecture. The master node is, in our scenario, responsible to assign a container-based analytics application to one of the available nodes in the corresponding cluster. Containerized applications are typically using Docker, a container platform used to build and isolate applications with a relevant stack of services. Here, to deploy an analytics application to edge nodes, a docker image has to be included in the Kubernetes manifest, i.e., deployment file, typically defined in YAML (see Fig. 6).

6.2 Target Application

We consider as our target application object detection, a typical video analytics processing in which an input set of video frames is analyzed. Analytics output is a list of detected objects with confidence levels and their positions on the image. We assume that edge keeps only a limited number of frames, e.g., sampling an industry-standard frame rate of 30fps and filtering only frames with significant changes or object movements, due to the limited capacity of edge nodes and efficient bandwidth usage [10].

In this experimental setup, we used the computation logic from our real-world application InTraSafEd 5G, used to perform object detection analytics to increase traffic and pedestrian safety with edge and 5G in the city of Vienna. The application runs a quantized version of SSD MobileNet v2 model [31], a lightweight and pre-trained convolutional neural network (CNN) based object detection. We dockerized the object detection logic and expose it as a service running in a container. Docker images for all node types, with and without edge TPU attached (Coral USB Accelerator enabling high-performance neural network inference), are available on the Docker hub repository[1], while the SEA-LEAP implementation is accessible on the GitHub repository[2]. Further, we used a PostgreSQL database running in a docker container to store metadata, i.e., location-related details about existing datasets.

6.3 Input Datasets

We evaluate proposed approaches using datasets typically used in computer vision analytics applications such as object detection and recognition [25]. To perform a complete evaluation, we select datasets of a different average size of image files, which allows having a wide diversity in terms of resolution, dimensions, and color depth. The main characteristics of datasets are presented in Table 3. For each dataset, we show the average size-frame ratio (γ) calculated as $\gamma(d) = size(d)/no\_frames(d)$, impacting SEA-LEAP placement optimizer (explained in Subsection 7.2).

Dataset *Intrasafed* comes from the InTraSafEd 5G project, containing sampled video frames from the chosen Vienna's intersection used for the real-time detection of critical situations and to support drivers in avoiding accidents. The frames are taken by traffic cameras and show critical situations where objects like pedestrians, cyclists, and pets, can appear in drivers' blind spots when turning on intersections.

Dataset *Penn-Fudan* comes from an image database used for object detection and recognition on areas around the University of Pennsylvania and Fudan University [32]. Selected frames represent various image qualities and angles of captured objects (pedestrians, bikes, and cars).

Datasets *Sherbrooke* and *René-Lévesque* come from the cameras monitoring different intersections, used for detecting and tracking multiple objects of various types in outdoor urban traffic surveillance [33]. Selected image frames represent different camera angles and resolutions, namely, a low camera monitoring cars, trucks, and pedestrians moving at an intersection (*Sherbrooke*) and a high camera covering three intersections with cars and bikes (*René-Lévesque*).

1. https://hub.docker.com/r/ilujic/inference-arm32v7/
2. https://github.com/lujic/sea-leap

TABLE 3
Main Characteristics of Datasets.

Dataset name	Frames	Size [MB]	Size/frame [MB]	Dimensions
Intrasafed	600	91.4	0.15	1280x720
Penn-Fudan	60	25.2	0.42	various
Sherbrooke	1800	154	0.09	800x600
René-Lévesque	3600	1011.8	0.28	1280x720

6.4 Testbed Configuration

In the experimental setup, we emulated a real-world system from our InTraSafEd 5G project, where node communication is handled by the MQTT broker, communicating to distant edge nodes deployed on traffic lights near a short-range cellular base station. Since latency is a critical requirement for our scenario, we evaluated the latency of deploying the broker either at the edge (inside the TU Wien's infrastructure) or using cloud service (hosted on MyQttHub). Network latency evaluation is shown in Fig. 4. We can see that edge deployment significantly reduces the latency (by 14.01%, 70.18%, 83.04% on average for 3G, 4G and 5G, respectively), making the edge meta-server placement as the best option for our setup.

Emulating and extending this real-world scenario, Fig. 5 shows our testbed configuration as well as an initial setup based on different edge sites (E1-E5). Edge sites represent small cells in a cellular network, featuring short-radius coverage of a small cell base station (as used in [30]), providing specific network connection types. Each site can contain one or multiple edge clusters, where in our setup contains multi-node (i.e., E1 and E3 including 3-node clusters, E2 including 2-node cluster) and single-node (i.e., E4 and E5) Kubernetes clusters. Edge meta-server represents a more reliable node (e.g., edge micro data center), able to communicate with edge nodes within different sites. Meta-scheduler receives from a user the description of an analytics request with a specific dataset.

We evaluate our emulation-based approach with the containerized analytics application, where as a baseline,

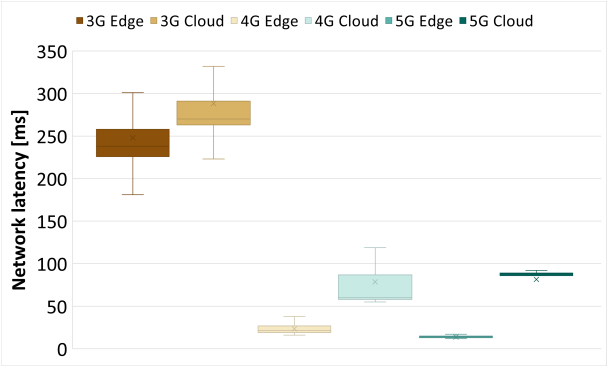
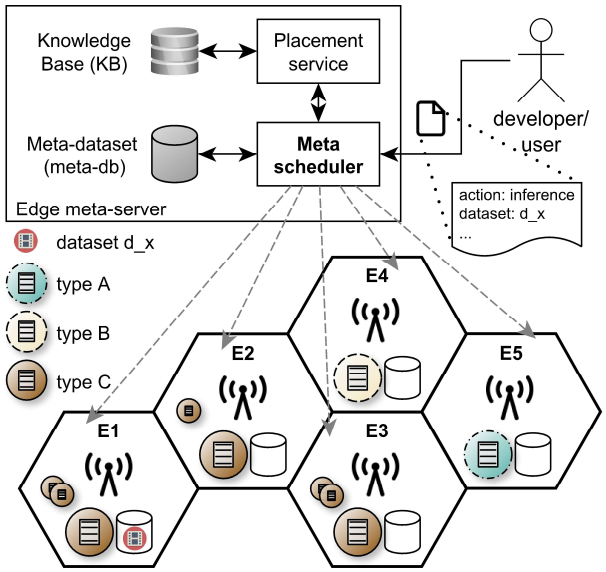


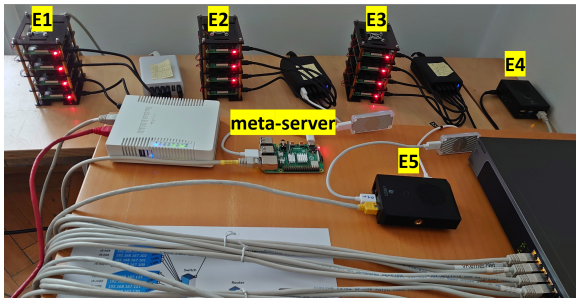
Fig. 4. Network latency for cloud/edge meta-server placement.

TABLE 4
Network latency and bandwidth benchmark (Vienna's suburb).

Network type	Latency [ms]	Bandwidth [Mbps]
3G	247.92	8.81
4G	23.44	41.43
5G	13.83	66.55



(a) The emulation-based scenario and network topology.



(b) Edge infrastructure overview.

Fig. 5. SEA-LEAP testbed configuration.

we measured inference times on the real-world datasets using our physical edge nodes, as described in Table 2. For each node type, we show the average inference time per single image. Results are averaged over 100 image frames for statistical significance since by adding more images the differences in inference time show a deviation of 39μs on average. These results are saved in the KB, which is used by Algorithm 3 for selecting the node which minimizes the latency of edge analytics placement. Further, Table 4 shows different latency and bandwidth measurements obtained using standard *iperf* application. The representative values are weighted averages of bandwidth collected on different network types in a suburb area of Vienna from the InTraSafEd 5G project and will be used in our placement optimizer (Section 7.2).

7 SEA-LEAP Evaluation

7.1 Static Placement Evaluation

Based on the *GuideMe* action (Section 5), the SEA-LEAP placement optimizer will enable the execution of the user's request on a node storing the required dataset, i.e., without considering alternative candidates (considered in Section 7.2). In the case of multiple locations storing the dataset, a node showing better performances (node type with a lower inference benchmark observation) will be prioritized. Otherwise, the algorithm will randomly select one of them. However, to enable the analytics execution on a target node using the requested dataset, the meta-scheduler needs to add a set of placement-specific details into a Kubernetes deployment file.

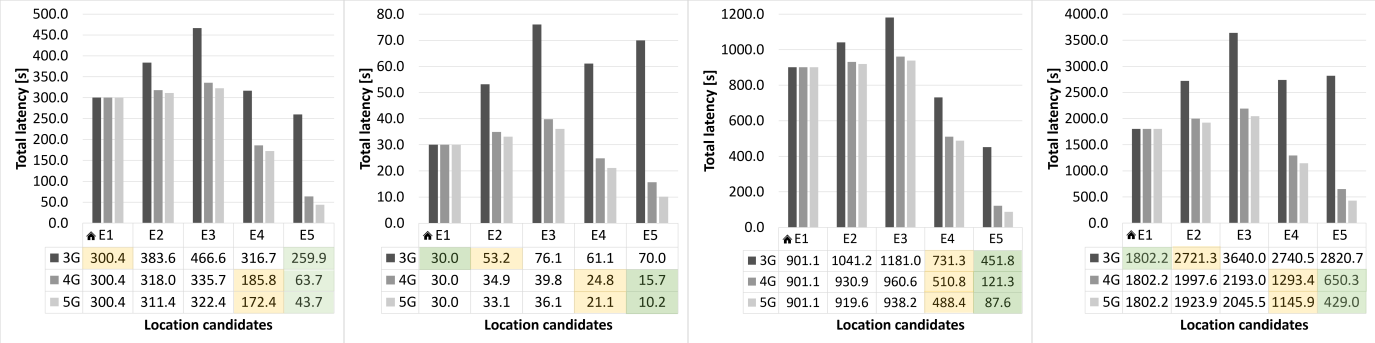
In our design, the meta-scheduler already stores different deployment templates that will be adapted with a specific set of information such as the node location, appropriate container image of the analytics application, and the dataset path on the target node (using *hostPath* as a volume). A simple example of an adapted deployment file is showed in Fig. 6. Based on specific lines from this description (i.e., the one including keyword *nodeName*), a distant master node will know where to place the analytics application in its cluster using the default scheduler. Beforehand, the edge meta-server is created as a single-node Kubernetes cluster and enabled to communicate to multiple clusters, using so-called Kubernetes configuration files with needed details (e.g., IP addresses of master nodes from our testbed edge sites). Followed by a meta-scheduler command to process a certain input dataset on the exposed analytics application, the obtained results can be forwarded back to a user. That said, the proposed SEA-LEAP meta-scheduler is designed as a new service that can be used on top of existing schedulers as a feature in different edge scenarios that require data locality-aware analytics placement.

7.2 Adaptive Data Movement Evaluation

In this experiment, we want to evaluate the *FollowMe* action (Section 5). SEA-LEAP placement optimizer will consider alternative location candidates different than the initial node storing the required dataset, and select the option with the lowest total latency. Thus, the placement algorithm estimates the total latency (based on details from KB) including the transfer of requested data from the source to an alternative location. Fig. 7 shows the results of the SEA-LEAP placement optimizer applied to each dataset from Table 3. In this representative example, the source node location of a dataset is set to the edge site E1. For that reason, the source location from E1 represents at the same time an initial candidate for analytics placement. Other alternative candidates will include additional network latency due to

```
...
spec:
  containers:
  - name: inference
    image: inference-notpu:latest
    ports:
    - containerPort: 5000
    volumeMounts:
    - mountPath: /data
      name: input-data
  volumes:
  - name: input-data
    hostPath:
      path: /data
      type: Directory
  nodeName: E1-m01
```

Fig. 6. SEA-LEAP deployment YAML file example.



(a) Intrasafed (b) Penn-Fudan (c) Sherbrooke (d) René-Lévesque

Fig. 7. SEA-LEAP placement calculation of node location candidates in different edge sites, driven by *GuideMe* and *FollowMe* actions. It is based on network connection benchmarks from a real-world edge location. For all cases, the source location of the dataset is set to E1.

needed dataset transfer. Green and yellow shaded locations show first and second-best candidates, respectively.

For dataset *Intrasafed* (Fig. 7(a)), the selected appropriate node location for analytics placement in each network type results in moving the dataset from the source node. In all cases of 3G (low), 4G (medium), and 5G (high) network conditions, we can decrease the total latency by 13.47%, 78.81%, and 85.46%, respectively, by moving the dataset to a candidate location in E5. For dataset *Penn-Fudan* (Fig. 7(b)), in low network conditions, the selected appropriate node location for analytics placement will be the source location, while for medium and high network conditions the total latency becomes 47.77% and 66.14% lower by moving dataset. Also, even in scenarios where the most appropriate location cannot host the analytics application or the dataset (e.g., because of limited capacity, high failure probability), selecting the second-best candidate (E4) can achieve 17.44% and 29.70% lower total latency for medium and high network conditions, respectively. For dataset *Sherbrooke* (Fig. 7(c)), for all network types, moving dataset can bring 49.86% (3G), 86.54% (4G), and 90.28% (5G) lower latency than in the source location initially storing the dataset. The dataset *René-Lévesque* (Fig. 7(d)) with the largest number of frames and overall size can benefit from better network conditions, achieving 63.92% and 76.20% lower total latency for medium and high bandwidth availability, respectively.

To evaluate optimizer's applicability to near real-time systems, we measure its runtime, included in the total latency, averaged over 100 times for statistical significance. We observe an average runtime of $1.41ms$ for the real testbed (Fig. 5). We also evaluate average runtime by increasing number of candidate nodes up to 100, resulting in $13.89ms$ and $26.29ms$ respectively with 1 or 2 source node locations.

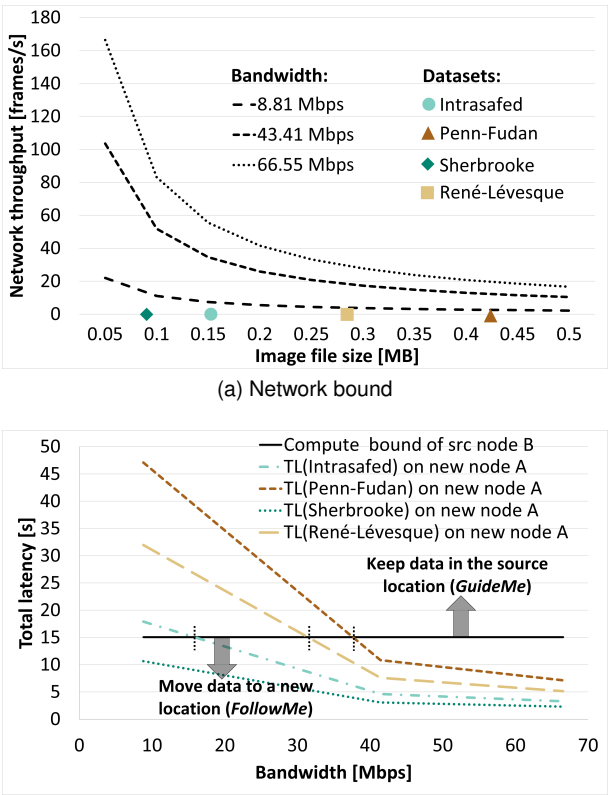
7.3 Discussion

Results show benefits of SEA-LEAP, i.e., it allows (i) autonomous placement of analytics requests, and (ii) the self-adaptation to data locality by considering both network and node candidate characteristics. For example, although node types A and B have respectively 28x and 2x lower inference time per frame compared to the source node type C (see Fig. 5), not all datasets benefit from their movements if available bandwidth is low.

Despite edge sites' different network characteristics, the network performance depends on the network bound of the node storing the dataset. As described in Section 5.2, the total latency of placing analytics to new locations is also impacted by other factors, i.e., network latency, number of hops, analytics execution time, and dataset size. Still, based on experimental results, in specific cases moving the dataset allows an average total latency reduction by 65.85%. Fig. 8 shows the SEA-LEAP placement decision rule and which aspects mostly have impact on whether to move data close to deployed on-demand analytics or vice versa. The estimated total latency of analytics placement is largely affected by two main aspects, namely, network bound (available bandwidth) and compute (node performance) bound.

Fig. 8(a) shows the relation between the average image file size and network throughput. We see that to a higher bandwidth corresponds a higher network bound for transferring images from a specific dataset, but does not hold for the compute-bound of a specific node candidate. This is because the target inference application resizes each input frame to the same dimension due to performance reasons. Since resizing has no effect on object detection accuracy, also average inference per frame (i.e., compute-bound) is unaffected. Consequently, computation time per node type depends exclusively on the number of input image frames.

For example, in Fig. 8(b), the number of input image frames for each dataset is 60, while the initial node candidate is a source node type B, and only nodes with lower inference time per frame (i.e., type A) are evaluated as alternative nodes. The solid black line shows the compute-bound of the source node B as the baseline, i.e., the total latency of running requested analytics on the dataset in the source node is equal to $\sim 15s$ ($60 \cdot 250.74ms$). Dashed lines show the estimated total latency of running analytics in alternative nodes, including data transfer over different network characteristics with two hops. With the available bandwidth in the source node, the placement optimizer decides whether to place analytics to (i) initial dataset location (*GuideMe*) for all results above the baseline, or (ii) a new node to which the dataset is moved (*FollowMe*) for all results below the baseline. Our solution shows that considering data locality in edge analytics placement can significantly improve overall analytics requests execution time.



(a) Network bound

(b) Placement decision

Fig. 8. SEA-LEAP placement decision. Subfigure (a) shows network bounds for various image file sizes. Subfigure (b) shows a borderline of placing analytics between the source node B and a new node A, among different datasets (60 frames) and available bandwidths with two hops.

7.4 Assumptions and Limitations

The SEA-LEAP placement optimizer estimates total latency for initial and alternative locations based on prior obtained edge metrics such as network characteristics (e.g., network types and number of hops between edge sites) and analytics benchmarks on node types. Also, we assume in our setup that edge nodes have access to existing docker images of analytics applications. Still, we partially address these issues by designing SEA-LEAP parts as generic services, which can be easily extended to consider (i) other analytics applications and datasets (e.g., critical time series analytics for failure prevention in smart manufacturing), (ii) different network topologies and characteristics, and (iii) conditions for filtering location candidates for analytics placement.

In the proposed solution, a centralized edge meta-server represents a single point of failure, which could affect SEA-LEAP reliability. Also, we assume a network of edge servers managed by trusted infrastructure providers controlling access to edge computing resources. Data security and privacy issues are delegated to the trusted infrastructure. Even though accessing metadata via the meta-scheduler or through the tracking mechanism already provides access control, additional protection measures could be taken in security-critical scenarios (e.g., in use cases with sensitive information). Lastly, we focus on data locality, while resource allocation is handled by Kubernetes. SEA-LEAP can be integrated on top of existing systems such as Kubernetes, facilitating data locality-aware edge analytics placement.

8 Related Work

Analytics placement and data management. Analytics placement at the edge has been discussed in several recent works. In [12], the authors propose a service-oriented resource management framework for fog computing focusing on service reliability. The paper [13] proposes EdgeEye, a service enabling the development and execution of video analytics applications. EdgeBox [11] is an architecture to improve automatic event detection in edge near real-time video analytics. However, these works limit placement to a specific cloud/edge location. Authors in [25] discuss the decentralized and federated edge infrastructure, focusing on a scalable approach to perform data collection and video analytics at the edge. Still, these approaches do not consider data locality and adaptive analytics placement. Current data management approaches adopt storage services configured toward centralized data aggregation [34] or geo-distributed data storage [35]. The work [36] surveys existing solutions for IoT data management. In [10], the relationship between resource availability and accuracy of edge-cloud video analytics are investigated, without considering data locality. In [19], Firework system is described facilitating distributed data processing requests, considering only specific locations.

Latency-aware scheduling and data locality. The work [37] addressed the offloading of computation-intensive tasks on edge nodes as an optimization problem. The proposed scheduling approach minimizes latency by static offloading of dependent tasks according to input data. In [14] latency-aware placement of data stream analytics applications is proposed, while [15] performs low-latency data management over geo-distributed and heterogeneous edge infrastructures. We focus on a latency-aware placement of on-demand analytics using data locality. The exploitation of data locality has been considered by other works in literature. For example, [20] discuss the concept of Semantic Cache, which employs a caching technique for edge analytics while reducing latency compared to cloud-only inference. In [38], the spatio-temporal locality of analytics is used to improve workload balancing between edge and cloud servers. Other works for analytics placement exploit data locality considering the edge-cloud workload balance perspectives [16], the trade-off between the resource usage and query accuracy [17], or the fairness of cloud resource allocation [18]. However, these works either do not consider the autonomous placement of on-demand analytics or focus on different aspects than minimizing requests execution time. We bridge these gaps by considering data locality in the self-adaptive placement of on-demand edge analytics.

Placement decision. The service placement decisions and trade-offs between local execution and remote execution are discussed in voluntary-based computing environments [39] and micro-cloud infrastructures [40]. Further, [41] proposes replica management and replica selection strategies in data grids, based on data mining techniques. Furthermore, concerning network service chaining at the edge, [42] looked at latency-aware service execution through software-defined approaches. Most of these placement decisions are based on either network-performance or resource allocation aspects. In this paper, we target data locality-aware, generic and self-adaptive mechanism that facilitates the edge analyt-

ics deployment across different edge infrastructures, while minimizing overall execution time for on-demand analytics.

Big edge data analytics. In [43], big data processing at the edge is discussed from the point of view of energy-efficient edge scheduling and impact of energy on QoS. [44] established a framework for offloading tasks to the edge, focusing mostly on minimizing the delay and cost of the computation. In [45], the bandwidth-adjustment problem in video-streaming is addressed, proposing a framework to reduce network traffic and adapt to conditions of mobile users. Other works address caching of mobile big data traffic at the edge [46] or enable federated query evaluations across cloud and fog nodes to reduce communication [47], but data locality-aware analytics placement is not considered.

9 Conclusions and Future Work

Executing on-demand edge analytics brings critical challenges to (i) identify locations of requested input datasets, and (ii) determine the target computational node where analytics must be deployed to minimize the overall completion time of user analytics requests. We propose SEA-LEAP (Self-adaptive and Locality-aware Edge Analytics Placement) framework to address the aforementioned issues.

SEA-LEAP includes a tracking mechanism for event-triggered data management and registration of data movements. On top of it, we propose a generic control mechanism featuring a meta-scheduler and placement optimizer. Our solution allows self-adaptive, on-the-fly placement of on-demand analytics based on data locality, and minimizes overall request execution time by performing adaptive data movements. We evaluate SEA-LEAP by considering on-demand video analytics application using our physical edge infrastructure and benchmarks. Results show benefits for users and developers, automating the placement of analytics requests and reducing the total latency by 65.85% on average for certain network and node characteristics. We believe that SEA-LEAP is a valuable step towards data locality-aware placements of on-demand analytics for edge multi-cluster or hybrid environments.

In the future, we plan to consider the single point of failure of the meta-scheduler, proposing a solution to improve scalability and reliability of meta-scheduler, i.e., by using multiple instances and implementing replication strategies of meta-dataset. We also plan to investigate the scalability of the tracking mechanism by experimenting different edge storage technologies such as Ceph, Minio, or other object storage technologies. Finally, we plan to further investigate the privacy and the data protection of SEA-LEAP.

Acknowledgments

The work described in this paper has been partially funded through the Rucon project (Runtime Control in Multi Clouds), FWF Y 904 START-Programm 2015, 5G Use Case Challenge InTraSafEd 5G (Increasing Traffic Safety with Edge and 5G) funded by the City of Vienna and supported through Ivan Lujic's netidee scholarship by the Internet Foundation Austria. Part of this research was carried out during Ivan Lujic's internship at IBM Research-Ireland.

References

[1] Q. Zhang, H. Sun, X. Wu, and H. Zhong, "Edge video analytics for public safety: A review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1675–1696, 2019.

[2] P. Patel, M. I. Ali, and A. Sheth, "On using the intelligent edge for iot analytics," *IEEE Intelligent Systems*, vol. 32, no. 5, pp. 64–69, 2017.

[3] Z. Ning, J. Huang, and X. Wang, "Vehicular fog computing: Enabling real-time traffic management for smart cities," *IEEE Wireless Communications*, vol. 26, no. 1, pp. 87–93, 2019.

[4] E. Ahmed, I. Yaqoob, I. A. T. Hashem, I. Khan, A. I. A. Ahmed, M. Imran, and A. V. Vasilakos, "The role of big data analytics in internet of things," *Computer Networks*, vol. 129, pp. 459–471, 2017.

[5] J. Ren, Y. Pan, A. Goscinski, and R. A. Beyah, "Edge computing for the internet of things," *IEEE Network*, vol. 32, no. 1, pp. 6–7, 2018.

[6] I. Lujic, V. De Maio, and I. Brandic, "Resilient edge data management framework," *IEEE Transactions on Services Computing*, vol. 13, no. 4, pp. 663–674, 2020.

[7] A. Aral and I. Brandic, "Learning spatiotemporal failure dependencies for resilient edge computing services," *IEEE Transactions on Parallel and Distributed Systems*, pp. 1–1, 2020.

[8] C. Wang, C. Gill, and C. Lu, "Adaptive data replication in real-time reliable edge computing for internet of things," in *2020 IEEE/ACM Fifth International Conference on Internet-of-Things Design and Implementation (IoTDI)*. IEEE, 2020, pp. 128–134.

[9] Q. Fan and N. Ansari, "Towards workload balancing in fog computing empowered iot," *IEEE Transactions on Network Science and Engineering*, 2018.

[10] G. Ananthanarayanan, P. Bahl, P. Bodík, K. Chintalapudi, M. Philipose, L. Ravindranath, and S. Sinha, "Real-time video analytics: The killer app for edge computing," *computer*, vol. 50, no. 10, pp. 58–67, 2017.

[11] B. Luo, S. Tan, Z. Yu, and W. Shi, "Edgebox: Live edge video analytics for near real-time event detection," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2018, pp. 347–348.

[12] M. Aazam and E.-N. Huh, "Dynamic resource provisioning through fog micro datacenter," in *2015 IEEE international conference on pervasive computing and communication workshops (PerCom workshops)*. IEEE, 2015, pp. 105–110.

[13] P. Liu, B. Qi, and S. Banerjee, "Edgeeye: An edge service framework for real-time intelligent video analytics," in *Proceedings of the 1st International Workshop on Edge Systems, Analytics and Networking*, 2018, pp. 1–6.

[14] A. da Silva Veith, M. D. de Assuncao, and L. Lefevre, "Latency-aware placement of data stream analytics on edge computing," in *International Conference on Service-Oriented Computing*. Springer, 2018, pp. 215–229.

[15] H. Gupta, Z. Xu, and U. Ramachandran, "Datafog: Towards a holistic data management platform for the iot age at the network edge," in {*USENIX*} *Workshop on Hot Topics in Edge Computing (HotEdge 18)*, 2018.

[16] Y. Guo, S. Wang, A. Zhou, J. Xu, J. Yuan, and C.-H. Hsu, "User allocation-aware edge cloud placement in mobile edge computing," *Software: Practice and Experience*, 2019.

[17] C.-C. Hung, G. Ananthanarayanan, P. Bodik, L. Golubchik, M. Yu, P. Bahl, and M. Philipose, "Videoedge: Processing camera streams using hierarchical clusters," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2018, pp. 115–131.

[18] J. Ru, Y. Yang, J. Grundy, J. Keung, and L. Hao, "An efficient deadline constrained and data locality aware dynamic scheduling framework for multitenancy clouds," *Concurrency and Computation: Practice and Experience*, p. e6037, 2020.

[19] Q. Zhang, Q. Zhang, W. Shi, and H. Zhong, "Firework: Data processing and sharing for hybrid cloud-edge analytics," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 9, pp. 2004–2017, 2018.

[20] S. Venugopal, M. Gazzetti, Y. Gkoufas, and K. Katrinis, "Shadow puppets: Cloud-level accurate {AI} inference at the speed and economy of edge," in {*USENIX*} *Workshop on Hot Topics in Edge Computing (HotEdge 18)*, 2018.

[21] I. Lujic, V. D. Maio, K. Pollhammer, I. Bodrozic, J. Lasic, and I. Brandic, "Increasing traffic safety with real-time edge analytics and 5g," in *Proceedings of the 4th International Workshop on Edge Systems, Analytics and Networking*, 2021, pp. 19–24.

[22] B. Cheng, A. Papageorgiou, and M. Bauer, "Geelytics: Enabling on-demand edge analytics over scoped data sources," in *IEEE International Congress on Big Data*, 2016, pp. 101–108.

[23] J. Santos, T. Wauters, B. Volckaert, and F. De Turck, "Towards network-aware resource provisioning in kubernetes for fog computing applications," in *2019 IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2019, pp. 351–359.

[24] P.-H. Tsai, H.-J. Hong, A.-C. Cheng, and C.-H. Hsu, "Distributed analytics in fog computing platforms using tensorflow and kubernetes," in *2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS)*. IEEE, 2017, pp. 145–150.

[25] M. Satyanarayanan, P. Simoens, Y. Xiao, P. Pillai, Z. Chen, K. Ha, W. Hu, and B. Amos, "Edge analytics in the internet of things," *IEEE Pervasive Computing*, vol. 14, no. 2, pp. 24–31, 2015.

[26] M. Breitbach, D. Schäfer, J. Edinger, and C. Becker, "Context-aware data and task placement in edge computing environments," in *2019 IEEE International Conference on Pervasive Computing and Communications (PerCom*. IEEE, 2019, pp. 1–10.

[27] C. Li, J. Tang, H. Tang, and Y. Luo, "Collaborative cache allocation and task scheduling for data-intensive applications in edge computing environment," *Future Generation Computer Systems*, vol. 95, pp. 249–264, 2019.

[28] H. B. Pasandi and T. Nadeem, "Convince: Collaborative cross-camera video analytics at the edge," in *2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. IEEE, 2020, pp. 1–5.

[29] T. Rausch, S. Nastic, and S. Dustdar, "Emma: Distributed qos-aware mqtt middleware for edge computing applications," in *2018 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2018, pp. 191–197.

[30] V. De Maio and I. Brandic, "Multi-objective mobile edge provisioning in small cell clouds," in *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*, 2019, p. 127–138.

[31] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[32] L. Wang, J. Shi, G. Song, and I.-F. Shen, "Object detection combining recognition and segmentation," in *Asian conference on computer vision*. Springer, 2007, pp. 189–199.

[33] J.-P. Jodoin, G.-A. Bilodeau, and N. Saunier, "Urban tracker: Multiple object tracking in urban mixed traffic," in *IEEE Winter Conference on Applications of Computer Vision*, 2014, pp. 885–892.

[34] B. Guidi and L. Ricci, "Aggregation techniques for the internet of things: An overview," in *The Internet of Things for Smart Urban Ecosystems*. Springer, 2019, pp. 151–176.

[35] V. Moysiadis, P. Sarigiannidis, and I. Moscholios, "Towards distributed data management in fog computing," *Wireless Communications and Mobile Computing*, vol. 2018, 2018.

[36] B. Diène, J. J. Rodrigues, O. Diallo, E. H. M. Ndoye, and V. V. Korotaev, "Data management techniques for internet of things," *Mechanical Systems and Signal Processing*, vol. 138, p. 106564, 2020.

[37] S. Yi, Z. Hao, Q. Zhang, Q. Zhang, W. Shi, and Q. Li, "Lavea: Latency-aware video analytics on edge computing platform," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, 2017, pp. 2573–2574.

[38] U. Drolia, K. Guo, J. Tan, R. Gandhi, and P. Narasimhan, "Cachier: Edge-caching for recognition applications," in *2017 IEEE 37th international conference on distributed computing systems (ICDCS)*. IEEE, 2017, pp. 276–286.

[39] B. Ali, M. A. Pasha, S. ul Islam, H. Song, and R. Buyya, "A volunteer-supported fog computing environment for delay-sensitive iot applications," *IEEE Internet of Things Journal*, vol. 8, no. 5, pp. 3822–3830, 2020.

[40] M. Selimi, L. Cerdà-Alabern, M. Sánchez-Artigas, F. Freitag, and L. Veiga, "Practical service placement approach for microservices architecture," in *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2017, pp. 401–410.

[41] T. Hamrouni, S. Slimani, and F. B. Charrada, "A survey of dynamic replication and replica selection strategies based on data mining techniques in data grids," *Engineering Applications of Artificial Intelligence*, vol. 48, pp. 140–158, 2016.

[42] P. Kathiravelu, P. Van Roy, and L. Veiga, "Composing network service chains at the edge: A resilient and adaptive software-defined approach," *Transactions on Emerging Telecommunications Technologies*, vol. 29, no. 11, p. e3489, 2018.

[43] Q. Zhang, M. Lin, L. T. Yang, Z. Chen, S. U. Khan, and P. Li, "A double deep q-learning model for energy-efficient edge scheduling," *IEEE Transactions on Services Computing*, vol. 12, no. 5, pp. 739–749, 2018.

[44] Q. Luo, C. Li, T. Luan, and W. Shi, "Minimizing the delay and cost of computation offloading for vehicular edge computing," *IEEE Transactions on Services Computing*, 2021.

[45] D. Wang, Y. Peng, X. Ma, W. Ding, H. Jiang, F. Chen, and J. Liu, "Adaptive wireless video streaming based on edge computing: Opportunities and approaches," *IEEE Transactions on services Computing*, vol. 12, no. 5, pp. 685–697, 2018.

[46] Y. Liu, Q. He, D. Zheng, X. Xia, F. Chen, and B. Zhang, "Data caching optimization in the edge computing environment," *IEEE Transactions on Services Computing*, 2020.

[47] M. Malensek, S. L. Pallickara, and S. Pallickara, "Hermes: Federating fog and cloud domains to support query evaluations in continuous sensing environments," *IEEE Cloud Computing*, vol. 4, no. 2, pp. 54–62, 2017.

Ivan Lujic is a PhD candidate at the Institute of Information Systems Engineering of the Vienna University of Technology, Austria. In 2018 he received a scholarship by netidee, Austria's largest Internet funding initiative, by the Internet Foundation Austria (IPA). He received his master's degree in Computer Science in 2016 from the University of Split, Croatia. His main research interests are data management strategies for near real-time Cloud/Edge analytics.

Vincenzo De Maio received his PhD in 2016 at the University of Innsbruck, Austria. His research in the area of parallel and distributed systems comprises energy-aware Cloud computing and scheduling. Since 2017, he is a postdoctoral researcher at the Institute of Information Systems Engineering of the Vienna University of Technology. He authored different conference and journal publications on the topic of energy efficiency and modeling for Cloud and Edge computing.

Srikumar Venugopal is a Research Scientist in IBM Research Europe. His research interests lie in the area of large-scale distributed systems, specifically in the topics of elasticity, resource management, cloud middleware, and data-intensive computing. He has published over 50 refereed papers and has held academic positions at UNSW Australia and the University of Melbourne. He obtained his Ph.D. from University of Melbourne in 2006.

Ivona Brandic is a Professor at Vienna University of Technology. In 2015 she was awarded FWF START prize, the highest Austrian award for young researchers. She received her PhD degree in 2007 from Vienna University of Technology. In 2011 she received the Distinguished Young Scientist Award from the Vienna University of Technology for her project on the Holistic Energy Efficient Hybrid Clouds. Her main research interests are cloud computing, large scale distributed systems, energy efficiency, QoS and autonomic computing.