




# Lapse: Latency & Power-Aware Placement of Data Stream Applications on Edge Computing

Carlos Henrique Kayser<sup>1</sup><sup>a</sup>, Marcos Dias de Assunção<sup>2</sup><sup>b</sup> and Tiago Ferreto<sup>1</sup><sup>c</sup>

<sup>1</sup>*School of Technology, Pontifical Catholic University of Rio Grande do Sul, Porto Alegre, Brazil*

<sup>2</sup>*Dept. of Software Engineering and IT, École de Technologie Supérieure, Univ. of Quebec, Montreal, Canada*

**Keywords:** Stream Processing, Edge Computing, Scheduling, Power Consumption, Service Level Agreement (SLA).

**Abstract:** Data Stream Processing (DSP) systems have gained considerable attention in edge computing environments to handle data streams from diverse sources, notably IoT devices, in real-time at the network's edge. However, their effective utilization concerning end-to-end processing latency, SLA violations, and infrastructure power consumption in heterogeneous environments depends on the adopted placement strategy, posing a significant challenge. This paper introduces *Lapse*, an innovative cost-based heuristic algorithm specifically crafted to optimize the placement of DSP applications within edge computing environments. *Lapse* aims to concurrently minimize latency SLA violations and curtail the overall power consumption of the underlying infrastructure. Simulation-driven experiments indicate that *Lapse* outperforms baseline strategies, substantially reducing the power consumption of the infrastructure by up to 24.42% and SLA violations by up to 75%.

## 1 INTRODUCTION


The emergence of the Internet of Things (IoT) led to an explosion in the number of devices generating large amounts of data at the edge of the network (Gubbi et al., 2013), enabling a new set of low-latency applications in areas like fraud detection, marketing, advertising, and public safety (Satyanarayanan et al., 2015). Take, for instance, a surveillance video application tasked with analyzing footage from multiple cameras across a metropolitan region to detect an accident, manage a disaster, or locate a missing person in real-time. While this process generates substantial data at the network edge, transmitting the footage from numerous cameras to the cloud for processing and analysis may be impractical, primarily due to the immense network bandwidth required (Satyanarayanan et al., 2015).


Edge computing emerged as an alternative to overcome these issues by enabling processing near where the data is generated (Satyanarayanan, 2017). Data Stream Processing (DSP) systems are popular in this field given their distributed nature and ability to process data as it is generated (Arkian et al., 2020)


(Cardellini et al., 2019). However, scheduling DSP applications on heterogeneous infrastructure is not a trivial task, and doing so generally requires coping with problems known to be NP-hard (Benoit et al., 2013) (Cardellini et al., 2016).

Given the heterogeneous nature of edge computing infrastructures, where some hosts may consume more energy than others and use intermittent and renewable energy sources, application placement becomes a critical decision with a dual role (Materwala et al., 2022) (Souza et al., 2023a). Firstly, it directly impacts the latency delivered to end-users based on the network location and the performance capabilities of edge servers. Secondly, it influences the overall energy consumption of the edge infrastructure. Strategic application placement strategies are necessary to balance latency and energy-saving goals. Moreover, it is crucial to establish mechanisms to ensure consistent performance of DSP applications to meet user requirements. Failure to meet these expectations can be costly, resulting in Service Level Agreement (SLA) violations and financial losses for infrastructure providers (Xu et al., 2021).

Previous work on scheduling DSP applications on edge computing considered mainly their end-to-end processing latency (da Silva Veith et al., 2018) (da Silva Veith et al., 2021) (Xu et al., 2022) or infrastructure power consumption (Loukopoulos et al.,

<sup>a</sup> <https://orcid.org/0000-0001-5459-2134>

<sup>b</sup> <https://orcid.org/0000-0002-4218-0260>

<sup>c</sup> <https://orcid.org/0000-0001-8485-529X>

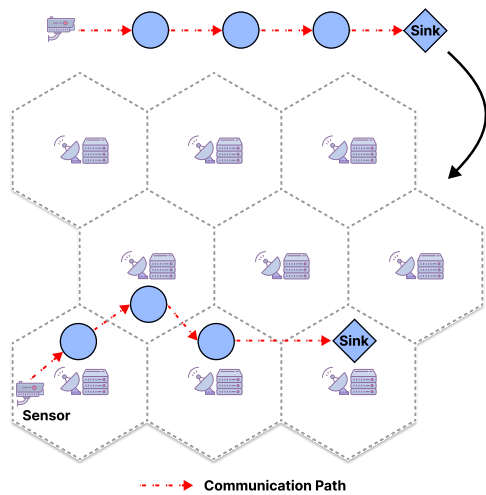


Figure 1: Placement of a DSP application's operators.

2018b) (Loukopoulos et al., 2018a). However, as far as we know, no work jointly optimizes the SLA violations incurred by excessive end-to-end processing latency and the infrastructure's energy consumption. This paper presents *Lapse*, a cost-based heuristic algorithm that schedules DSP applications on edge infrastructure, jointly optimizing their end-to-end processing latency SLA violations and overall infrastructure power consumption.

The rest of this paper is structured as follows. Section 2 presents an overview of the main concepts considered in this work. Next, Section 3 discusses the related work and highlights our contributions. Section 4 and 5 present our system model and solution. Section 6 presents our performance evaluation methodology and results. Finally, Section 7 concludes the paper and delivers final remarks.

## 2 BACKGROUND

Edge computing (Satyanarayanan et al., 2009) is a distributed computing paradigm that places computing resources physically closer to the end-user and smart devices to improve the reliability, security, privacy, and latency of cloud-centric deployments (Shi et al., 2016). In addition, edge computing minimizes the amount of data uploaded to the cloud for processing, allowing local data processing. This synergizes with DSP, a distributed computing paradigm that allows real-time, high-volume, heterogeneous, and continuous processing of data streams, enabling a large set of applications such as intelligent traffic management, real-time surveillance, and smart grids (Andrade et al., 2014).

Existing DSP engines, such as Apache Storm<sup>1</sup> and Apache Flink<sup>2</sup> represent applications as Directed Acyclic Graphs (DAGs), where the vertices represent operators that execute specific actions (e.g., filter, group by, count, etc) onto the incoming data, and the edges represent how the data flows between operators (Xu et al., 2022). The data streams enter the application through a data source operator, flow to the downstream operators for processing, and so on until they reach a data sink, which can be a database or a message system such as Apache Kafka<sup>3</sup>.

Most DSP applications are latency sensitive, requiring a response as quickly as possible to satisfy the performance requirements regarding latency and throughput (Andrade et al., 2014). Latency represents the time spent processing the data from the data source operator to the data sink operator. In contrast, throughput refers to the rate at which the system processes the data streams. Scheduling DSP applications onto highly distributed environments such as edge computing can be challenging (Arkian et al., 2020) since network latency, available bandwidth, and computing capacity can vary, impacting the application latency and throughput.

Figure 1 illustrates the scheduling of a real-time surveillance application with four operators processing the data streams from a surveillance camera on an edge computing infrastructure of nine servers, each in a region represented by a hexagon. The application's sensor and sink locations are fixed within the infrastructure as the sensor monitors a specific geographical area. In contrast, the sink represents the destination for processed data, often decided a priori. The produced data is generally critical for systems that rely on it for various purposes (Xu et al., 2022) (Cardellini et al., 2016).

## 3 RELATED WORK

The problem of placing DSP applications onto computing resources has been studied extensively in the literature. (Cardellini et al., 2016) proposed an optimal placement strategy, treating it as an Integer Linear Programming (ILP) problem and optimizing Quality of Service (QoS) metrics like latency and availability, considering the heterogeneity of the edge computing infrastructure. They integrated this model into Apache Storm to place DSP application operators. (Hiessl et al., 2019) argue that the approach neglected

<sup>1</sup><https://storm.apache.org/>

<sup>2</sup><https://flink.apache.org/>

<sup>3</sup><https://kafka.apache.org/>

leased resources and migration costs, introducing an extended ILP formulation considering these factors. However, neither work addresses the optimization of power consumption on edge servers, a critical concern in edge computing infrastructure (Materwala et al., 2022).

The study conducted by (Loukopoulos et al., 2018b) aims to reduce energy consumption and average delay for end-users deploying DSP applications at the edge of the Internet. They introduce two algorithms: one prioritizes minimizing delay, while the other addresses both delay and energy consumption. Additionally, (Loukopoulos et al., 2018a) presents another contribution, namely Pareto-Efficient Algorithm (PEA), which tackles the same problem.

In (da Silva Veith et al., 2018), strategies were proposed to optimize the placement of DSP applications, across cloud and edge computing infrastructures. The goal was to enhance application response time by considering pre-defined data source and sink locations within the infrastructure. Similarly, (da Silva Veith et al., 2021) argued that existing approaches lack scalability for extensive cloud-edge infrastructure with numerous IoT devices. Consequently, the authors introduced three scalable placement strategies for DSP applications, prioritizing end-to-end latency minimization.

In another study, (Xu et al., 2022), Amnis was introduced as a solution to minimize end-to-end latency for DSP applications in edge computing environments. Their approach incorporates data locality awareness and resource constraints to improve physical plan generation and operator placement. Previous investigations have examined the placement problem considering several factors, such as end-to-end latency, power consumption, and availability. However, to our knowledge, no prior work has proposed placement solutions that simultaneously address end-to-end processing latency SLA violations while minimizing power consumption in edge servers.

## 4 SYSTEM MODEL

This section presents the system model for the DSP application operator placement problem in edge computing environments. First, we describe the main attributes and behavior of each entity, covering the operator placement process and the application performance computation. Then, we introduce our optimization objectives. Table 1 summarizes the notation used in the rest of this work.

We consider an edge computing infrastructure that extends the cellular network (Klas, 2017), providing

Table 1: Summary of the notation used in this paper.

| Notation  | Description   |
|---|---|
| $\mathcal{E}$   | Set of edge servers   |
| $\mathcal{B}$   | Set of base stations  |
| $\mathcal{L}$   | Set of network links  |
| $\mathcal{S}$   | Set of sensors  |
| $\mathcal{A}$   | Set of applications   |
| $\mathcal{O}$   | Set of operators  |
| $\mathcal{N}$   | Set of network switches   |
| $e_f$   | $\mathcal{L}_f$ 's communication latency (ms)                           |
| $b_f$   | $\mathcal{L}_f$ 's bandwidth (Gbps)                                     |
| $c_i$   | $\mathcal{E}_i$ 's CPU capacity (MIPS)                                  |
| $r_i$   | $\mathcal{E}_i$ 's RAM capacity (GB)                                    |
| $q_i$   | $\mathcal{E}_i$ 's static power consumption (watts)                     |
| $m_i$   | $\mathcal{E}_i$ 's max power consumption (watts)                        |
| $u_j$   | $\mathcal{A}_j$ 's operator chain                                       |
| $g_j$   | $\mathcal{A}_j$ 's processing latency SLA by event (ms)                 |
| $t_j$   | $\mathcal{A}_j$ 's sensor   |
| $h_k$   | $\mathcal{O}_k$ 's MIPS demand to process one event                     |
| $d_k$   | $\mathcal{O}_k$ 's RAM demand (MB)                                      |
| $\lambda_k$   | $\mathcal{O}_k$ 's input event rate                                     |
| $\pi_k$   | $\mathcal{O}_k$ 's input event size (bytes)                             |
| $x_{i,k}$   | Operator placement matrix   |
| $\mu(\mathcal{E}_i)$                                      | $\mathcal{E}_i$ 's utilization percentage                               |
| $\xi(\mathcal{E}_i)$                                      | $\mathcal{E}_i$ 's power consumption (watts)                            |
| $\rho(\mathcal{A}_j, \mathcal{O}_k)$                      | Returns the previous operator or data source                            |
| $\tau(\rho(\mathcal{A}_j, \mathcal{O}_k), \mathcal{O}_k)$ | Time to transfer the data stream to operator $\mathcal{O}_k$ (sec.)     |
| $\sigma(\mathcal{O}_k, \mathcal{E}_i)$                    | $\mathcal{O}_k$ 's processing rate at edge server $\mathcal{E}_i$       |
| $\Upsilon(\mathcal{O}_k, \mathcal{E}_i)$                  | Time to operator $\mathcal{O}_k$ process data at $\mathcal{E}_i$ (sec.) |
| $P(\mathcal{A}_j)$  | $\mathcal{A}_j$ 's end-to-end processing latency (sec.)                 |
| $\Theta(u_{j,k-1}, u_{j,k})$                              | Cumulative delay between elements (ms)                                  |
| $\Omega(\mathcal{E}_{i-1}, \mathcal{E}_i)$                | The bandwidth available between elements (Gbps)                         |

access to a set of edge servers  $\mathcal{E}$  through base stations  $\mathcal{B}$ . A group of hexagonal cells represents the infrastructure topology as in (Aral et al., 2021), where each cell provides wireless access to users through a radio base station. In each hexagonal cell, there is a network switch ( $\mathcal{N}$ ) that interconnects the base stations and edge servers to the neighboring hexagons via a set of network links  $\mathcal{L}$ . Hence,  $\mathcal{L}_f = \{e_f, b_f\}$  represents a network link, where  $e_f$  is the network latency (expressed in milliseconds), and  $b_f$  is the network link bandwidth capacity depicted as gigabits per second (Gbps).

An edge server is represented as  $\mathcal{E}_i = \{c_i, r_i, q_i, m_i\}$ , where  $c_i$  and  $r_i$  represent the CPU capacity in Millions of Instructions per Second (MIPS) and RAM capacity in gigabytes (GB). In addition,  $q_i$  and  $m_i$  are the static power consumption (i.e., the amount of power the machine consumes in idle) and maximum power consumption of edge server  $\mathcal{E}_i$  defined in watts. We consider a set of devices  $\mathcal{S}$ , such as sensors and cameras, strategically positioned at the edge infrastructure. They produce data streams processed by a set of DSP applications  $\mathcal{A}$  composed of a set of operators  $\mathcal{O}$ . Each DSP application is a DAG, where the nodes represent the operators, and the edges represent the data flow between them. An application is represented as  $\mathcal{A}_j = \{u_j, g_j, t_j\}$ , where  $u_j$  represents a list of operator chain, for example  $u_j = \{O_1, O_2, O_3\}$ . Let  $g_j$  be the application end-to-end processing latency SLA by event (i.e.,

application's latency threshold), and  $t_j$  the application data source  $S_j$ .

An operator  $O_k = \{h_k, d_k, \lambda_k, \pi_k\}$ , poses a demand of  $h_k$  MIPS to process one event, and requires  $d_k$  of RAM capacity expressed as megabytes (MB).  $O_k$ 's input event rate and input event size (bytes) are given by, respectively,  $\lambda_k$  and  $\pi_k$ . The applications' sinks – i.e., the last downstream operator(s) – are pinned to the infrastructure as detailed in Section 2. As edge servers host the operators, we represent the operator placement scheme through a matrix  $x_{i,k}$ , where:

$$x_{i,k} = \begin{cases} 1 & \text{if edge server } \mathcal{E}_i \text{ hosts operator } O_k \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

The edge server  $\mathcal{E}_i$ 's utilization percentage is computed considering the operators  $O_k$  demand and the edge server  $\mathcal{E}_i$  capacity  $c_i$ , according to Equation 2 as in (Loukopoulos et al., 2018b) (Loukopoulos et al., 2018a). We also consider that the edge server's power consumption, denoted by the function  $\xi(\mathcal{E}_i)$ , is determined by the ratio of MIPS capacity utilization and demand (Equation 3), as per the same references.

$$\mu(\mathcal{E}_i) = \frac{\sum_{k=1}^{|\mathcal{O}|} x_{i,k} \cdot h_k \cdot \lambda_k}{c_i} \quad (2)$$

$$\xi(\mathcal{E}_i) = q_i + \mu(\mathcal{E}_i) \cdot (m_i - q_i) \quad (3)$$

Equation 5 determines how much time in seconds is needed to transfer the data stream from the data source (e.g., a previous operator  $O_{k-1}$  or a sensor  $S_j$  indicated by Equation 4) to operator  $O_k$ . More precisely, the transfer time mainly depends on the amount of data stream ( $\lambda_k \cdot \pi_k$ ), and  $\Omega(\rho(\mathcal{A}_j, O_k), O_k)$  which denotes the available bandwidth between them.

The bandwidth available to transfer the data streams between edge servers (e.g.,  $\mathcal{E}_1, \mathcal{E}_2$ ) is given by  $\Omega(\mathcal{E}_1, \mathcal{E}_2)$ , that is the minimum bandwidth available between the network links  $\mathcal{L}$  that interconnect  $\mathcal{E}_1$  and  $\mathcal{E}_2$ . We also consider an equal share policy to schedule the data streams in the network infrastructure (i.e., the network link bandwidth capacity is divided equally between the data streams). Furthermore, the network delay between edge servers (e.g.,  $\mathcal{E}_1, \mathcal{E}_2$ ) or operators (e.g.,  $O_1, O_2$ ) is given by  $\Theta(O_{k-1}, O_k)$ . Both functions use Dijkstra's Shortest Path Algorithm (Dijkstra et al., 1959) with the network links latency  $e_f$  as weight.

$$\rho(\mathcal{A}_j, O_k) = \begin{cases} t_j & \text{if } O_k = A_{j,0} \\ O_{k-1} & \text{otherwise.} \end{cases} \quad (4)$$

$$\tau(\rho(\mathcal{A}_j, O_k), O_k) = \frac{\lambda_k \cdot \pi_k}{\Omega(\rho(\mathcal{A}_j, O_k), O_k) + \Theta(\rho(\mathcal{A}_j, O_k), O_k) * \lambda_k} \quad (5)$$

The processing rate of an operator  $O_k$  placed on an edge server  $\mathcal{E}_i$  is represented by  $\sigma(O_k, \mathcal{E}_i)$ , according to Equation 6. With that, we can determine how much time in seconds is needed for the operator  $O_k$  to process the data streams on the edge server  $\mathcal{E}_i$ , which is denoted by the function  $\Upsilon(O_k, \mathcal{E}_i)$  (Eq. 7). We assume that the edge server  $\mathcal{E}_i$  MIPS capacity  $c_i$  is divided equally between the operators hosted. With that, we can simulate the impact of consolidating many operators on the same host.

$$\sigma(O_k, \mathcal{E}_i) = \frac{c_i}{h_k \cdot \sum_{k=1}^{|\mathcal{O}|} x_{i,k}} \quad (6)$$

$$\Upsilon(O_k, \mathcal{E}_i) = \frac{\lambda_k}{\sigma(O_k, \mathcal{E}_i)} \quad (7)$$

The end-to-end processing latency is the sum of network latency, data transmission, and processing time required by the operators to process the data streams on the host edge server. More formally, the end-to-end processing latency is denoted by the function  $P(\mathcal{A}_j)$  (Eq. 8). For each operator  $u_{j,k}$  of an application  $\mathcal{A}_j$ , we compute the time to transfer the data stream between it and the previous operator with  $\tau(\rho(\mathcal{A}_j, u_{j,k}), u_{j,k})$ . Furthermore, we compute the processing time for all edge servers  $\mathcal{E}_i$  with an operator  $u_{j,k}$ . Then, we divide the results by  $\lambda_k$ , the operator  $u_{j,k}$  input event rate to obtain the latency processing per event. We consider the transfer cost between operators hosted on the same edge server negligible.

$$P(\mathcal{A}_j) = \sum_{k=1}^{|\mathcal{A}_j|} \frac{(\tau(\rho(\mathcal{A}_j, u_{j,k}), u_{j,k}) + \sum_{i=1}^{|\mathcal{E}|} x_{i,k} \cdot \Upsilon(u_{j,k}, \mathcal{E}_i))}{\lambda_k} \quad (8)$$

This work aims to establish the placement of DSP applications' operators to minimize the applications' end-to-end processing latency SLA violations and overall power consumption of edge servers. More formally, Equation 9 represents the objective function, subject to three constraints. The first constraint, Equation 10, guarantees that each operator is placed in just one edge server. Since edge servers do not have unlimited capacity, Equation 11 guarantees that the placement of operators respects the edge server MIPS capacity, and Equation 12, ensures that the operators' demands do not exceed the edge server RAM capacity, as in (da Silva Veith et al., 2021).

$$\text{Minimize } \sum_{j=1}^{|\mathcal{A}|} [P(\mathcal{A}_j) > g_j] + \sum_{i=1}^{|\mathcal{E}|} \xi(\mathcal{E}_i) \quad (9)$$

**Subject to:**

$$\sum_{i=1}^{|\mathcal{E}|} x_{i,k} = 1, \forall k \in \{1, \dots, |\mathcal{O}|\} \quad (10)$$



$$\sum_{k=1}^{|\mathcal{O}|} h_k \cdot \lambda_k \cdot x_{i,k} \leq c_i, \forall i \in \{1, \dots, |\mathcal{E}|\} \quad (11)$$

$$\sum_{k=1}^{|\mathcal{O}|} (d_k + (\lambda_k \cdot \pi_k)) \cdot x_{i,k} \leq r_i, \forall i \in \{1, \dots, |\mathcal{E}|\} \quad (12)$$

## 5 PROPOSED STRATEGY

This section presents *Lapse*, our strategy to place DSP applications on edge computing environments, minimizing their end-to-end processing latency SLA violations and the edge servers' power consumption.

*Lapse* places DSP applications' operators using a Depth First Search (DFS) approach, i.e., it deploys all application operators before proceeding to the next application. It does that by prioritizing applications with greater demand (e.g., the sum of the demand of all its operators) and with lower end-to-end processing latency SLA (Alg. 2, line 2). Then, for each application  $\mathcal{A}_j \in \mathcal{A}'$ , *Lapse* identifies the location of the application's sensor  $\varphi$  and sink  $\eta$  (Alg. 2, lines 4-5), and finds the edge servers between them through Algorithm 1. To do so, Algorithm 1 starts by finding the shortest paths between the sensor and sink locations (Alg. 1, line 3). Then, for each path, the algorithm iterates over each network switch and its neighbors' network switches searching for edge servers (Alg. 1, lines 3-8). Finally, the algorithm returns the edge servers discovered along the path with the greatest number of such servers.

*Lapse* iterates over the list of application's operators  $u_j$  once the set of suitable edge servers  $\mathcal{E}'$  has been found. First, it checks whether an operator  $O_k$  is already assigned to a resource, as we consider that the placement of sink operators is determined beforehand. After that, *Lapse* sorts the set of edge servers  $\mathcal{E}'$  as per Equation 13, which considers the sum of the min-max normalization of i) aggregated network latency between  $\varphi$  and  $\mathcal{E}'$ ; ii) aggregated network latency between  $\eta$  and  $\mathcal{E}'$ ; and iii)  $\mathcal{E}'$  max power consumption ( $m_i$ ) (Alg. 2, line 11). Then, *Lapse* iterates over edge servers, checking for each if it can host the operator  $O_k$ . Upon finding a server, we update  $\varphi$  with the edge server  $\mathcal{E}'$ . Finally, regarding the case where not enough resources to host the operator  $O_k$  are available, we repeat the process considering all edge servers until the operator  $O_k$  is scheduled (Alg. 2, lines 17-18).

$$\text{norm}(\Theta(\varphi, \mathcal{E}_i)) + \text{norm}(\Theta(\eta, \mathcal{E}_i)) + \text{norm}(m_i) \quad (13)$$

Algorithm 1: Find possible edge servers.

---

**Input:**  $\varphi$  (Application's sensor location)  
**Input:**  $\eta$  (Application's sink location)  
**Output:**  $\mathcal{E}'$  (Set of edge servers)

```

1  $\varepsilon \leftarrow 0$ 
2  $\delta \leftarrow \text{null}$ 
3  $\zeta \leftarrow$  shortest paths between  $\varphi$  and  $\eta$ 
4 foreach path  $\zeta' \in \zeta$  do
5    $\mathcal{E}' \leftarrow \emptyset$ 
6   foreach network switch  $\mathcal{N}_v \in \zeta'$  do
7     if network switch  $\mathcal{N}_v$  has an edge server
8        $\mathcal{E}_i$  then
9          $\mathcal{E}' \leftarrow \mathcal{E}' \cup \{\mathcal{E}_i\}$ 
10      foreach neighbor network switch  $\mathcal{N}_v'$  of
11         $\mathcal{N}_v$  do
12          if network switch  $\mathcal{N}_v'$  has an edge
13            server  $\mathcal{E}_i$  then
14               $\mathcal{E}' \leftarrow \mathcal{E}' \cup \{\mathcal{E}_i\}$ 
15      if  $|\mathcal{E}'| > \varepsilon$  then
16         $\varepsilon \leftarrow |\mathcal{E}'|$ 
17         $\delta \leftarrow \mathcal{E}'$ 
18 return  $\delta$ 

```

---

## 6 PERFORMANCE EVALUATION

This section presents the methodology used to evaluate *Lapse*'s effectiveness in placing DSP applications on edge computing infrastructures.

### 6.1 Experiments Description

Our simulated edge computing infrastructure comprises 25 edge servers positioned at specific positions on the map defined by the K-means algorithm (MacQueen, 1967). To configure our set of edge servers, we consider three server specifications from (Ismail and Materwala, 2021), as shown in Table 2. The edge servers are interconnected by a partially connected hexagonal mesh network topology (Aral et al., 2021) with 208 network links  $\mathcal{L}$ , where each of them has 10 or 20 milliseconds of latency and 0.1 Gbps of bandwidth. All parameter values presented in this section are uniformly distributed unless otherwise indicated.

The scenario has 8 sensors, denoted as  $\mathcal{S}$ , responsible for generating data streams. A corresponding set of 8 applications, represented as  $\mathcal{A}$ , should process these data streams. We employ the K-means algorithm to uniformly position the sensors throughout the entire area and prevent scenarios where all sensors cluster around one location. Each application is associated with a distinct sensor, serving as its primary data source. We consider a linear application topology (that is, an application that does not have parallel operators), where each application may have several

Algorithm 2: Lapse algorithm.

```

1  $x_{i,k} \leftarrow 0$ 
2  $\mathcal{A}' \leftarrow$  Applications sorted by  $\sum_{k=1}^{|u_j|} h_k * \lambda_k$  and  $g_j$ 
   (desc.)
3 foreach application  $\mathcal{A}_j \in \mathcal{A}'$  do
4    $\varphi \leftarrow$  location of application's sensor
5    $\eta \leftarrow$  location of application's sink
6    $\mathcal{E}' \leftarrow$  find_edge_servers( $\varphi, \eta$ )
7   foreach operator  $O_k \in u_j$  do
8     if operator  $O_k$  is already placed then
9       continue
10    while operator  $O_k$  is not placed do
11       $\mathcal{E}' \leftarrow$  edge servers sorted by Eq. 13
        (desc.)
12      foreach  $\mathcal{E}_i \in \mathcal{E}'$  do
13        if  $\mathcal{E}_i$  has capacity to host  $O_k$ 
        (Eq. 11-12) then
14           $x_{i,k} \leftarrow 1$ 
15           $\varphi \leftarrow \mathcal{E}_i$ 
16          break
17        if operator  $O_k$  could not be placed
        then
18           $\mathcal{E}' \leftarrow \mathcal{E}$ 
19 return  $x_{i,k}$ 

```

Table 2: Specifications of edge servers (Ismail and Materwala, 2021).

| Model   | CPU       | RAM   | Power Idle | Power Max. |
|---------|-----------|-------|------------|------------|
| Model 1 | 2750 MIPS | 32 GB | 265 W      | 1387 W     |
| Model 2 | 3500 MIPS | 64 GB | 127 W      | 559 W      |
| Model 3 | 3000 MIPS | 64 GB | 45 W       | 276 W      |

operators varying from  $\{8, 12, 16\}$  operators, and an end-to-end processing latency SLA according to the specifications established by the 3rd Generation Partnership Project (3GPP) (3GPP, 2022), varying from 80 and 100 milliseconds to process a single event. Regarding the operators' specifications, the amounts of

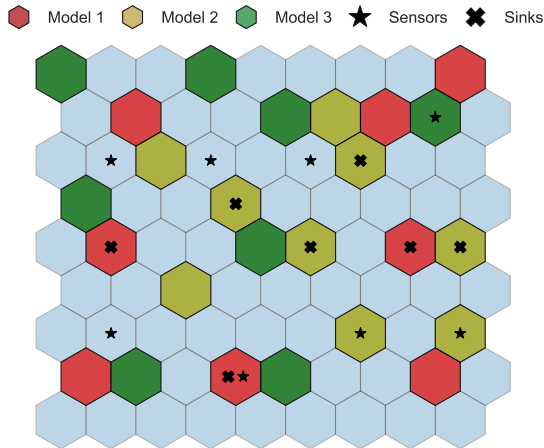


Figure 2: Edge computing infrastructure.

CPU and RAM resources are uniformly selected from the values listed in Table 3.

Furthermore, Algorithm 3 also randomly positions the applications' sink on the infrastructure. The sink placement algorithm randomly chooses an edge server to place the applications' sink by considering the edge servers around the sensor location. More precisely, the edge servers considered for the random choice are from one to three hops away from the application's sensor location (Algorithm 3, lines 7-8) to avoid placing applications' sinks too far from the sensor, which could cause SLA violations due to the distance.

Algorithm 3: Sink placement algorithm.

```

1 foreach application  $\mathcal{A}_j \in \mathcal{A}'$  do
2    $\varphi \leftarrow$  location of application's sensor
3    $\mathcal{E}' \leftarrow \emptyset$ 
4   foreach  $\mathcal{E}_i \in \mathcal{E}$  do
5     if  $\varphi$  is at same location of  $\mathcal{E}_i$  then
6       continue
7     if  $\mathcal{E}_i$  is at least 1 hop and no more than 3
        hops away from  $\varphi$  then
8        $\mathcal{E}' \leftarrow \mathcal{E}' \cup \{\mathcal{E}_i\}$ 
9    $\mathcal{E}' \leftarrow$  pick an edge server at random from  $\mathcal{E}'$ 
10   $x_{i,|u_j|-1} \leftarrow 1$ 

```

We consider multiple event sizes to simulate distinct types of data streams (e.g., text, image frames, voice recording (da Silva Veith et al., 2018)) that range from 10 KB, 50 KB, and 200 KB. In addition, the operators' input event rate is 5000 events per second. All operators within the same application have the same input event specification. Figure 2 illustrates the final scenario with an occupation level of approximately 50%.

Table 3: Operator demand configurations.

| Configuration | MIPS Demand | RAM Demand |
|---------------|-------------|------------|
| Spec. 1       | 0.03        | 70 MB      |
| Spec. 2       | 0.06        | 80 MB      |
| Spec. 3       | 0.09        | 90 MB      |
| Spec. 4       | 0.1         | 100 MB     |

To evaluate the performance of the proposed solution, *Lapse*, we extend the *EdgeSimPy* (Souza et al., 2023b) simulator by adding the system model presented in Section 4. *EdgeSimPy* is a simulation toolkit that enables modeling and evaluating resource management policies, such as placement and migration, in edge computing infrastructures.

We have compared *Lapse* with *AELS* (da Silva Veith et al., 2021), and Apache Storm's default resource-aware scheduler (*Storm*), which resembles a round-robin algorithm. *Storm*

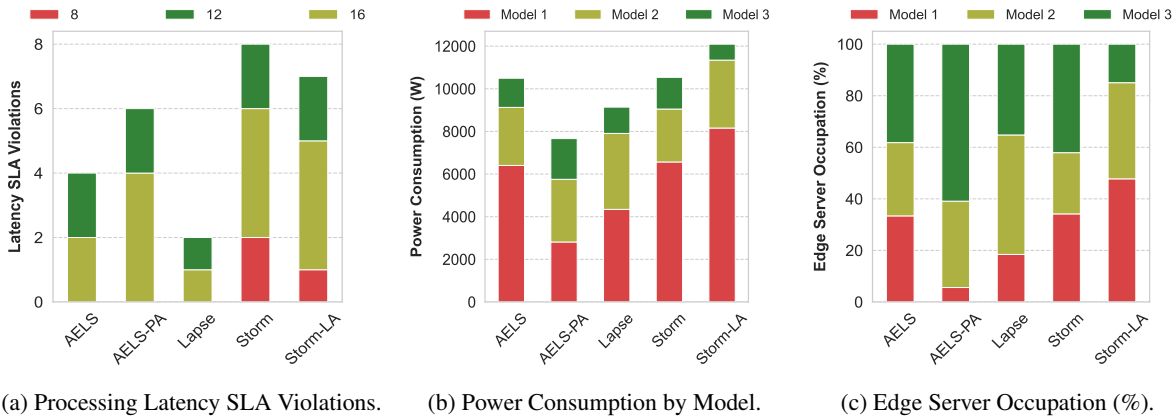


Figure 3: Final application placement of the various algorithms.

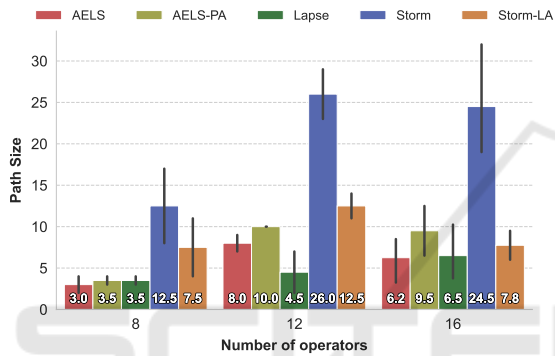


Figure 4: Application Path Size by Application Size.

starts by randomly selecting an edge server and placing as many operators as possible until the computational resources reach their limits. After saturation, it randomly chooses another server (Xu et al., 2022). Since **Storm** does not consider the edge server latency during the edge server selection, we have created a latency-aware version called **Storm-LA**, which selects the edge server with the lowest latency instead of randomly choosing an edge server.

In addition, since none of these algorithms consider infrastructure power consumption as an objective, we expand the **AELS** algorithm by introducing a power-aware version called **AELS-PA**, which considers both latency and edge servers' power consumption. Next, we compare and analyze the results regarding end-to-end processing SLA violations, and overall power consumption. To support reproducible research, we have made our source code public on our GitHub repository<sup>4</sup>.

## 6.2 Latency SLA Violations

Figure 3a shows the instances of end-to-end processing latency SLA violations across different application lengths, denoted by the number of operators, for each strategy used. As expected, the **Storm** strategy exhibited consistent SLA violations, as **Storm** does not account for application latency requirements. Consequently, it generates placements with lengthy application paths (Figure 4). This resulted in high network latency due to multiple inter-edge server communication. Furthermore, Figure 5 revealed that the data flows experienced a decrease in bandwidth due to competing resource allocation, leading to increased latency and incidence of SLA violations.

Despite **Storm-LA** considers the latency associated with the application's sensors, it could only mitigate a single SLA violation. In particular, **Storm-LA** exhibited a pronounced preference for *Model 1* servers, as shown in Figure 3c. These servers, characterized by their superior processing capabilities (as detailed in Table 2), facilitate the co-location of multiple operators on a single server, which reduces the amount of inter-node communication (Figure 4), effectively mitigating network traffic (Figure 5).

**AELS** strategy led to violations in four applications. In addition to selecting edge servers to deploy application operators based on latency estimation, which may yield suboptimal results due to the potential impact of unforeseen unprovisioned applications on network bandwidth availability, **AELS** also searches greedily for the best edge server to deploy the operator of each downstream application without considering the location of the sink, which could unexpectedly increase latency.

Contrarily, **AELS-PA** produced less favorable results when compared to the standard **AELS** algo-

<sup>4</sup><https://github.com/carloshkayser/lapse>

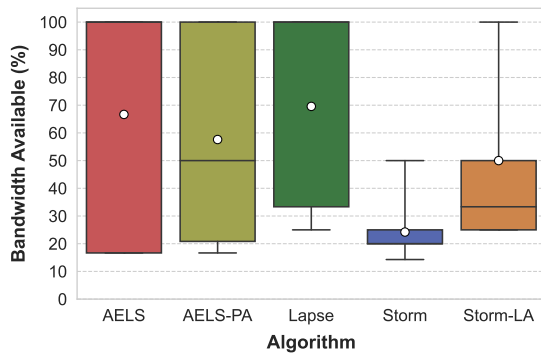


Figure 5: Bandwidth Available for each Data Flow.

algorithm, violating six applications, due to the conflicting objective of minimizing power consumption. As a result, **AELS-PA** preferred *Model 3* servers (Figure 3c), leading to an increase in the size of application paths (Figure 4).

On the contrary, **Lapse** achieved a notable reduction in SLA violations, with only two applications experiencing breaches. This result can be attributed, in part, to **Lapse** effectively minimizing the size of the application paths (Figure 4). This was possible due to consideration of the application's sink position and the granular selection of edge servers between the application's data source and sink. Furthermore, the allocation decisions made by **Lapse** proved to be highly efficient in terms of network utilization (Figure 5).

### 6.3 Overall Power Consumption

Figure 3b provides an overview of the power consumption in different edge server models. In contrast, Figure 3c illustrates the occupation of edge servers categorized by their respective models. Among the algorithms evaluated, **Storm** exhibited one of the highest power consumption levels, surpassing 10,000 watts, by not considering the power consumption of the edge servers.

In particular, **Storm-LA** further intensified power consumption, reaching 12,000 watts, due to the prioritization of low-latency edge servers, which led to the selection of *Model 1* edge servers instead of *Model 3*, as observed in Figure 3c compared to the baseline **Storm** algorithm.

Similarly, **AELS** approached a power consumption level of approximately 11,000 watts. Nevertheless, its power-aware version, **AELS-PA**, successfully reduced it to less than 8,000 watts. However, this improvement in power efficiency had repercussions on allocation decisions, particularly in terms of latency SLA violations.

**Lapse**, in contrast, achieved a significant reduction in power consumption, reducing it to approx-

imately 9,100 watts while minimizing the number of applications experiencing SLA violations. This achievement is due to **Lapse**'s preference for using edge servers belonging to *Model 2* and *Model 3*. These edge server models present a lower power consumption profile, a choice driven by the well-balanced cost function used by **Lapse**, which considers both latency and power consumption.

## 7 CONCLUSIONS

DSP systems have emerged in edge computing environments to process data streams in real-time close to the data sources, but their placement is not trivial due to its impact on user satisfaction, affecting end-to-end processing latency SLA violations and infrastructure power consumption. This paper introduces **Lapse**, a cost-based heuristic algorithm designed to address this problem by optimizing DSP application placement in heterogeneous edge computing infrastructures. Simulation-driven experiments showed that **Lapse** can reduce the power consumption of the infrastructure by up to 24.42% and SLA violations by up to 75%. In future work, we intend to evaluate **Lapse** in diverse scenarios and levels of occupancy, as well as explore a recursive algorithm inspired by the branch and bound technique (Levitin, 2005).

## ACKNOWLEDGEMENTS

This work was supported by the PDTI Program, funded by Dell Computadores do Brasil Ltda (Law 8.248 / 91). The authors acknowledge the High-Performance Computing Laboratory of the Pontifical Catholic University of Rio Grande do Sul for providing resources for this project.

## REFERENCES

- 3GPP (2022). Service requirements for the 5g system. *3rd Generation Partnership Project (3GPP), Technical Specification (TS) 22.261*.
- Andrade, H. C., Gedik, B., and Turaga, D. S. (2014). *Fundamentals of stream processing: application design, systems, and analytics*. Cambridge University Press.
- Aral, A., De Maio, V., and Brandic, I. (2021). Ares: Reliable and sustainable edge provisioning for wireless sensor networks. *IEEE Transactions on Sustainable Computing*, 7(4):761–773.
- Arkian, H., Pierre, G., Tordsson, J., and Elmroth, E. (2020). An experiment-driven performance model of stream



- processing operators in fog computing environments. In *Symposium on Applied Computing*, pages 1763–1771. ACM.
- Benoit, A., Dobrila, A., Nicod, J.-M., and Philippe, L. (2013). Scheduling linear chain streaming applications on heterogeneous systems with failures. *Future Generation Computer Systems*, 29(5):1140–1151.
- Cardellini, V., Grassi, V., Lo Presti, F., and Nardelli, M. (2016). Optimal operator placement for distributed stream processing applications. In *Distributed and Event-based Systems*, pages 69–80. ACM.
- Cardellini, V., Mencagli, G., Talia, D., and Torquati, M. (2019). New landscapes of the data stream processing in the era of fog computing.
- da Silva Veith, A., de Assuncao, M. D., and Lefevre, L. (2018). Latency-aware placement of data stream analytics on edge computing. In *Service-Oriented Computing (ICSOC)*, pages 215–229. Springer.
- da Silva Veith, A., de Assuncao, M. D., and Lefevre, L. (2021). Latency-aware strategies for deploying data stream processing applications on large cloud-edge infrastructure. *Transactions on Cloud Computing*.
- Dijkstra, E. W. et al. (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271.
- Gubbi, J., Buyya, R., Marusic, S., and Palaniswami, M. (2013). Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7):1645–1660.
- Hiessl, T., Karagiannis, V., Hochreiner, C., Schulte, S., and Nardelli, M. (2019). Optimal placement of stream processing operators in the fog. In *Fog and Edge Computing (ICFEC)*, pages 1–10. IEEE.
- Ismail, L. and Materwala, H. (2021). Escove: Energy-sla-aware edge-cloud computation offloading in vehicular networks. *Sensors*, 21(15).
- Klas, G. (2017). Edge computing and the role of cellular networks. *Computer*, 50(10):40–49.
- Levitin, A. (2005). The design and analysis of algorithms.
- Loukopoulos, T., Tziritas, N., Koziri, M., Stamoulis, G., and Khan, S. U. (2018a). A pareto-efficient algorithm for data stream processing at network edges. In *Cloud Computing Technology and Science (Cloud-com)*, pages 159–162. IEEE.
- Loukopoulos, T., Tziritas, N., Koziri, M., Stamoulis, G., Khan, S. U., Xu, C.-Z., and Zomaya, A. Y. (2018b). Data stream processing at network edges. In *International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 657–665. IEEE.
- MacQueen, J. (1967). Classification and analysis of multivariate observations. In *5th Berkeley Symp. Math. Statist. Probability*, pages 281–297.
- Materwala, H., Ismail, L., Shubair, R. M., and Buyya, R. (2022). Energy-sla-aware genetic algorithm for edge-cloud integrated computation offloading in vehicular networks. *Future Generation Computer Systems*, 135:205–222.
- Satyanarayanan, M. (2017). The emergence of edge computing. *Computer*, 50(1):30–39.
- Satyanarayanan, M., Bahl, P., Caceres, R., and Davies, N. (2009). The case for vm-based cloudlets in mobile computing. *Pervasive Computing*, 8(4):14–23.
- Satyanarayanan, M., Simoens, P., Xiao, Y., Pillai, P., Chen, Z., Ha, K., Hu, W., and Amos, B. (2015). Edge analytics in the internet of things. *Pervasive Computing*, 14(2):24–31.
- Shi, W., Cao, J., Zhang, Q., Li, Y., and Xu, L. (2016). Edge computing: Vision and challenges. *Internet of Things Journal*, 3(5):637–646.
- Souza, P., Kayser, C., Roges, L., and Ferreto, T. (2023a). Thea-a qos, privacy, and power-aware algorithm for placing applications on federated edges. In *Parallel, Distributed and Network-Based Processing (PDP)*, pages 136–143. IEEE.
- Souza, P. S., Ferreto, T., and Calheiros, R. N. (2023b). Edgesimpy: Python-based modeling and simulation of edge computing resource management policies. *Future Generation Computer Systems*.
- Xu, J., Palanisamy, B., Wang, Q., Ludwig, H., and Gopisetty, S. (2022). Amnis: Optimized stream processing for edge computing. *Journal of Parallel and Distributed Computing*, 160:49–64.
- Xu, L., Venkataraman, S., Gupta, I., Mai, L., and Potharaju, R. (2021). Move fast and meet deadlines: Fine-grained real-time stream processing with cameo. In *Symposium on Networked Systems Design and Implementation (NSDI)*, pages 389–405. USENIX.