Latency-Aware Strategies for Deploying Data Stream Processing Applications on Large Cloud-Edge Infrastructure

Alexandre da Silva Veith, Marcos Dias de Assuncao, Laurent Lefèvre

Latency-Aware Strategies for Deploying Data Stream Processing Applications on Large Cloud-Edge Infrastructure

Alexandre da Silva Veith, Marcos Dias de Assunção, Laurent Lefèvre

Abstract—Internet of Things (IoT) applications often require the processing of data streams generated by devices dispersed over a large geographical area. Traditionally, these data streams are forwarded to a distant cloud for processing, thus resulting in high application end-to-end latency. Recent work explores the combination of resources located in clouds and at the edges of the Internet, called cloud-edge infrastructure, for deploying Data Stream Processing (DSP) applications. Most previous work, however, fails to scale to very large IoT settings. This paper introduces deployment strategies for the placement of DSP applications onto cloud-edge infrastructure. The strategies split an application graph into regions and consider regions with stringent time requirements for edge placement. The proposed Aggregate End-to-End Latency Strategy with Region Patterns and Latency Awareness (AELS+RP+LA) decreases the number of evaluated resources when computing an operator's placement by considering the communication overhead across computing resources. Simulation results show that, unlike the state-of-the-art, AELS+RP+LA scales to environments with more than 100k resources with negligible impact on the application end-to-end latency.

Index Terms—Data Stream Processing, Edge Computing, Aggregate end-to-end latency, Operator placement

1 Introduction

The rapid growth of the Internet of Things (IoT) has led to scenarios where data is produced in overwhelming rates and which require (near) real-time analysis. In smart cities, IoT, and operational monitoring of large infrastructure, applications generate continuous data streams that must be processed under short delays. Streaming data is often gathered and analyzed by Data Stream Processing Engines (DSPEs) where an application is commonly structured as a dataflow. A dataflow comprises multiple sensors, gateways and/or actuators as *data sources* generating data; stateless or stateful *operators* such as filtering, and aggregation that perform transformations on the data; and *data sinks* or queries that consume or store the data. Traditionally, DSPEs place the whole dataflow on a single *cloud* [1].

The schedulers of DSPEs commonly assign tasks to resources in a round-robin fashion while ignoring intricacies of operators and resources. This often results in poor performance and load imbalance when considering the distributed nature of IoT. In IoT scenarios, data sources are mainly located at the edges of the Internet, and data is transferred to the cloud via long-distance links that increase the *end-to-end latency* (*i.e.*, the time data is generated to the time it reaches the sinks). The communication overhead incurred by transferring data via the Internet makes it hard to achieve near real-time processing on clouds alone. *Cloud-edge infrastructure* provides an alternative approach to deploy Data Stream Processing (DSP) applications by combining multiple cloud providers (*i.e.*, federated cloud) and resources at the edges of the Internet, often organized in edge sites.

Existing application placement strategies for cloud-edge infrastructure generally expect some sort of manual intervention [2] and many models are oblivious to complex operator transformation patterns (*e.g.*, stateful – store information about previous executions as new data is streamed in) [3] and do not support resource constraints (*e.g.*, memory and communication) [4] or dependencies between operators [5], [6]. Likewise, other efforts offer linear approaches [7] where the behavior of a DSP application deployment on heterogeneous infrastructure is non-stochastic. Another major drawback is that prior work fails to scale to cloud-edge infrastructure [8] where the number of edge resources can be in the hundreds or thousands. For instance, initiatives including the Humble Lamppost Project [9] and LinkNYC [10], and companies such as SMIGHT [11] and Edgemicro [12], illustrate that cloud-edge infrastructure will become even more pervasive and reach much larger scale in near future.

In this paper, we introduce a set of strategies for the initial placement of the operators onto federated cloud and edge resources while supporting large-scale infrastructure, considering the characteristics of resources and meeting application requirements. We consider synthetic and generic analytics applications with multiple sources and sinks distributed across cloud and edge resources. In particular, we decompose the application graph by identifying patterns such as the final destination of messages, and then placing operators across edge and cloud resources while considering their latency closeness. We also extrapolate the infrastructure topology size to show that the scalability of

- *A. da Silva Veith is with the University of Toronto. E-mail: alexandre.veith@utoronto.ca*
- *M. Dias de Assunção is with ETS Montreal. E-mail: Marcos.Dias-De-Assuncao@etsmtl.ca*
- *L. Lefèvre is with Inria, ENS of Lyon, University of Lyon. E-mail: laurent.lefevre@ens-lyon.fr*

Manuscript received in September, 2020

current approaches is an issue. The results demonstrate that our strategies are scalable and provide operator placements at a reasonable time. Comprehensive simulations comprising multiple network topologies and several application settings demonstrate the effectiveness of our approach in maintaining the end-to-end latency regardless of the network size. The results also show that our solution can cover large-scale infrastructure where the state-of-the-art fails.

The main motivation is to design operator placement for cloud-edge infrastructure that are scalable and do not penalize the end-to-end latency, which is a stringent requirement for near real-time analytics. The main contributions of this work are:

- It improves a model for DSP applications on heterogeneous infrastructure introduced in previous work [13] by extending it to address large-scale scenarios.
- Strategies for placing operators across large-scale network topologies.
- Evaluation of the strategies against traditional and state-of-the-art schemes.

The rest of this paper is organised as follows. Section 2 discusses related work. The application and infrastructure models are introduced in Section 3. The placement strategies are detailed in Section 4 whereas Section 5 presents performance evaluation results. Finally, Section 6 concludes the paper and discusses future work.

2 Related Work

Previous work reviewed strategies for operator placement in early distributed DSP systems [14]. Several approaches aim to optimize the application end-to-end latency by exploiting the location of computing resources in a network infrastructure, hence reducing inter-resource communication and maximizing processing throughput. Previous work, however, does not handle large IoT scenarios and is oblivious to the constraints of IoT resources regarding memory, CPU, and network bandwidth. Also, most solutions do not handle cloud-edge infrastructure, consider oversimplified DSP applications, and do not use IoT resources for DSP tasks.

Taneja's Cloud-Edge Placement (TCEP) [3] is a best-fit scheduling policy for application placement on cloud-edge infrastructure that organizes computing resources and application operators into two distinct vectors, which are then sorted by their CPU capacity and CPU requirement respectively. For each operator in the operators vector, TCEP evaluates whether the resource at the middle of the resources vector meets the operator's CPU, memory, and bandwidth requirements. TCEP ignores multiple or federated edge sites and the network latency, which results in high application end-to-end latency. It also does not estimate application characteristics, which is an issue when handling heterogeneous computing resources.

Redowan *et al.* [15] propose a policy where DSP operators are classified in a three-dimensional space (*i.e.*, user-defined end-to-end latency, amount of data per input and frequency for collecting data in IoT devices). Operators with stringent requirements are placed on the edge while non-stringent ones are assigned to the cloud. The work also offers an Integer Linear Programming (ILP) model for minimizing the application end-to-end latency, which has scalability issues, and considers scenarios with up to 25 devices. The work also ignores states when computing the end-to-end latency.

DROPLET [16] offers a scheduling policy by modeling the completion time of DSP applications as the shortest path problem. The proposed model ignores operators that change the pattern of input data, assuming equal sizes for output and input tuples. DROPLET considers only the operators' output queues and a single edge site. Moreover, memory and CPU capacity are ignored when assigning multiple operators to the same resource.

Cardellini *et al.* [7] introduce an ILP model for geo-distributed DSP systems that considers resource heterogeneity, but ignores memory constraints as it is conceived for federated clouds with large memory capacity. Edge computing resources, however, often have limited capacity and cannot host operators with heavy memory demands. Moreover, the solution has scalability issues as it relies on ILP. Communication-aware heuristics were proposed for assigning DSP operators to Virtual Machines (VMs) placed across federated clouds [17], [18], but these solutions do not consider edge computing resources and struggle to minimize the end-to-end latency since data sources are often located in IoT devices.

Canali *et al.* [19] introduce an ILP model and algorithm for placing applications onto cloud-edge infrastructure that focuses on reduce end-to-end latency from edge sites. They evaluate their solution by analyzing the traffic monitoring application of the Italian city of Modena, with a population of 180,000 inhabitants. Despite the significant population, the testbed was based on 89 sensors and seven cloud resources. Likewise, Geelytics [20] explores a cloud-edge infrastructure for minimizing the end-to-end latency of applications and respecting resource constraints. In contrast, Kiani *et al.* [21] attempt to minimize the energy-time cost using Mixed-Integer Nonlinear Programming (MINLP), but their solution handles only small network topologies, ignores multi-tenancy, and estimates the operator placement by considering all application operators and computing resources. Existing solutions do not address scenarios with a vast number of sensors, hence demonstrating that scalability is still an issue.

Table 1 summarizes how the related work handles the operator placement problem. Some existing work proposes solutions for federated clouds [17], [18], while ignoring IoT devices. Other approaches for stateful operators [7], [19] tackle cloud-edge infrastructure, but disregard memory and bandwidth constraints. TCEP [3] and Redowan *et al.* [15] consider some of these constraints, but their solutions do not support stateful operators, while Kiani *et al.* [21] restrict the scheduling of a single operator at the same edge location. Droplet [16] presents both a complexity analysis and experiments that demonstrate the efficiency of a two-layer architecture that considers a single cloud and one edge site. However, it is unclear whether the proposed architecture can scale to a scenario with multiple clouds and edge sites. Canali *et al.* [19] evaluate their approach in a large scenario, but ignore federated clouds. There is a lack of scalable solutions for cloud-edge infrastructure that support stateful operators, respect resource constraints and consider

TABLE 1
Current features in the related work.

Solutions	Infrastructure		Stateful Operators	Resource constraints			Scalable
	Cloud	Edge		Memory	CPU	Bandwidth	
TCEP [3]	✓	✓		✓	✓	✓	✓
Geelytics [20]	✓	✓			✓	✓	
Cardellini *et al.* [7]	✓	✓	✓		✓	✓	
Gu *et al.* [17]	✓		✓		✓	✓	
Chen *et al.* [18]	✓		✓		✓	✓	
Kiani *et al.* [21]	✓	✓			✓	✓	
Droplet [16]	✓	✓	✓			✓	
Canali *et al.* [19]	✓	✓	✓		✓		
Redowan *et al.* [15]	✓	✓		✓	✓	✓	
This work	✔	✔	✔	✔	✔	✔	✔

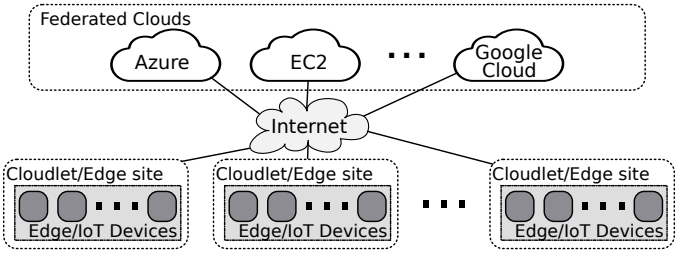


Fig. 1. Overview of cloud-edge infrastructure.

the scale of future IoT deployments [8].

3 System Model and Problem Definitions

This section describes the system model for operator placement introduced in our previous work [13], [22], extending it to cover stateful operators and to address scalability issues. Table 2 summarizes the symbols used in the paper.

3.1 System Model

Cloud-edge infrastructure comprises federated clouds and edge sites as depicted in Fig. 1. The clouds and edge sites are interconnected via the Internet. Each cloud resource represents a service supplied by a cloud service provider (*e.g.*, Amazon EC2 [23], Microsoft Azure [24], Google Cloud [25]). In contrast, each edge site has multiple and heterogeneous resources connected to a LAN. The internal communication of an edge site often adopts technologies such as LTE and other wireless equipment that increase the network latency significantly [26]. For this reason, the internal links and edge resources in an edge site are analyzed individually. This work considers that information on the network topology and resource availability are discovered and maintained via algorithms such as Vivaldi and Software Defined Network solutions [7].

We focus on application end-to-end latency as performance metric for placing DSP applications across edge and cloud resources. The DSP operator placement problem consists of accommodating the application components (*i.e.*, operators) onto the available infrastructure resources to optimize the end-to-end latency. Information about the application can be obtained via profiling techniques or from previous executions as demonstrated in existing work [27].

TABLE 2
Main symbols adopted for the problem description.

Symbol	Description
$\mathcal{R}$	Set of cloud ∪ edge resources
$\mathcal{L}$	Set of all network links
$\mathcal{N}, \mathcal{G}$	Network and application graphs
$k \leftrightarrow l$	A link connecting resources k and l
cpu_k^r, mem_k^r	CPU and memory capacities in MIPS and bytes of resource k
$lat_{k \leftrightarrow l}$,$bdw_{k \leftrightarrow l}$	Latency and bandwidth in seconds and bps of bidirectional link $k \leftrightarrow l$
$\mathcal{O}$	Set of operators
$\mathcal{O}^{src}$, $\mathcal{O}^{trn}$, $\mathcal{O}^{out}$	Set of data sources, transformations and data sink operators
$\mathcal{E}$	Set of event streams between operators
cpu_i^o, mem_i^o	CPU and memory req. of operator i
ws_i^o	Length of operator i's window in number of events
ψ_i^o	Selectivity of operator i
ω_i^o	Data compression rate of operator i
$e_{i \to j}$	Event stream between operator i and j
$e_{i \to j}^{\rho}$	Event stream with probability ρ that an event emitted by operator i will flow to j
$\lambda_i^{in}, \lambda_i^{out}$	Input/output event rate of operator i
$\varsigma_i^{in}, \varsigma_i^{out}$	Input/output event size of operator i
$stime_{\langle i,k \rangle}$	Service time in seconds of operator i at resource k
$ctime_{\langle i,k \rangle \langle j,l \rangle}$	Communication time in seconds from operator i at resource k to j at l
$mem_{\langle i,k \rangle}$	Overall memory required by operator i when deployed at resource k
p_i, L_{p_i}	A graph path and its end-to-end latency
$\mathcal{P}$	The set of all paths in an application graph
$\mu_{\langle i,k \rangle}$	The rate at which operator i can process events at resource k
$\mathbb{1}_{\langle e_{i \to j}, k \leftrightarrow l \rangle}$	Indicates when the stream between operators i and j has been assigned to the link between resources k and l
$\mathbb{1}_{\langle i,k \rangle}$	Indicates whether operator i is placed on resource k
AL	Aggregate end-to-end latency

The infrastructure is viewed as a graph $\mathcal{N} = (\mathcal{R}, \mathcal{L})$ where $\mathcal{R}$ is the union set of cloud compute resources and edge resources, and $\mathcal{L}$ is the set of logical links interconnecting the resources, comprising WAN interconnections and LAN links. A computational resource is defined as a tuple $r_k = \langle cpu_k^r, mem_k^r \rangle \in \mathcal{R}$, where cpu_k^r is the CPU capability

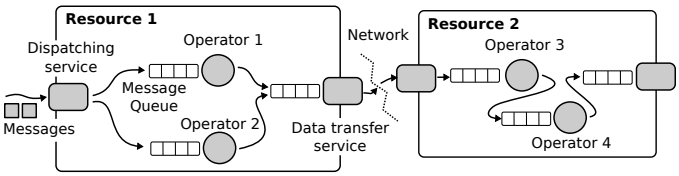


Fig. 2. Four operators and their queues placed on two resources.

in Millions of Instructions per Second (MIPS) and mem_k^r is the memory capability in bytes. Similarly, a network link is a tuple $k \leftrightarrow l = \langle bdw_{k\leftrightarrow l}, lat_{k\leftrightarrow l}\rangle \in \mathcal{L}$, where $k \leftrightarrow l$ represents the interconnection between resource k and l, $bdw_{k\leftrightarrow l}$ the bandwidth capability in bits per second (bps), and $lat_{k\leftrightarrow l}$ the latency in seconds. We consider the latency of a resource k to itself (*i.e.*, $lat_{k\leftrightarrow k}$) to be 0.

3.2 Application Model

A DSP application is a graph $\mathcal{G} = (\mathcal{O}, \mathcal{E})$ of *operators* $\mathcal{O}$ comprising *data sources* $\mathcal{O}^{src}$, *data sinks* $\mathcal{O}^{out}$ where data is stored or published, and *transformations* $\mathcal{O}^{trn}$ that execute functions over the incoming data, and streams $\mathcal{E}$ of data events flowing between operators. Each operator is a tuple $o_i = \langle cpu_i^o, mem_i^o, \psi_i^o, \omega_i^o, ws_i^o\rangle \in \mathcal{O}$, where cpu_i^o is the CPU requirement in MIPS to handle an individual event, mem_i^o is the memory requirement in bytes to load the operator in a computing resource, ψ_i^o is the ratio of number of input events to output events (*i.e.*, selectivity), ω_i^o is the ratio of the size of input events to the size of output events (*i.e.*, operator data transformation pattern), and ws_i^o is the length of the operator's window in number of events. The rate at which operator i can process events at resource k is denoted by $\mu_{\langle i,k\rangle}$ and is essentially $\mu_{\langle i,k\rangle} = cpu_k^r \div cpu_i^o$. An operator can have one or multiple output streams. An event stream $e_{i\to j}^{\rho} \in \mathcal{E}$ connects operator i to j with a probability ρ that an output event emitted by i will flow through to j. If $e_{i\to j}^{\rho}$ is the only output stream of operator i, then $e_{i\to j}^{\rho} \in \mathcal{E} = 1$.

The rate at which operator i produces events (λ_i^{out}) is a product of its input event rate λ_i^{in} and its selectivity ψ_i^o. The output event rate of a data source operator $z \in \mathcal{O}^{src}$ depends on the number of measurements it takes from a sensor or another monitored resource. We can then recursively compute the input and output event rates for a downstream operator j as follows:

$$\lambda_i^{in} = \lambda_z^{out} \qquad \forall e_{z\to i} \in \mathcal{E}, z \in \mathcal{O}^{src} \quad (1)$$

$$\lambda_j^{in} = \sum_{e_{i\to j}\in\mathcal{E}} \lambda_i^{in} \times \psi_i^o \times e_{i\to j}^{\rho} \qquad \forall i,j \in \mathcal{O}, i \notin \mathcal{O}^{src} \quad (2)$$

$$\lambda_j^{out} = \lambda_j^{in} \times \psi_j^o \qquad \forall j \in \mathcal{O}, j \notin \mathcal{O}^{out} \quad (3)$$

Likewise, we can recursively compute the average size ς_i^{in} of events that arrive at a downstream operator i and the size of events it emits ς_i^{out} by considering the upstream operators' event sizes and their respective operator data transformation pattern (*i.e.*, ω_i^o). In other words:

$$\varsigma_i^{in} = \varsigma_z^{out} \qquad \forall e_{z\to i} \in \mathcal{E}, z \in \mathcal{O}^{src} \quad (4)$$

$$\varsigma_j^{in} = \sum_{e_{i\to j}\in\mathcal{E}} \varsigma_i^{in} \times \omega_i^o \times e_{i\to j}^{\rho} \qquad \forall i,j \in \mathcal{O}, i \notin \mathcal{O}^{src} \quad (5)$$

$$\varsigma_j^{out} = \varsigma_j^{in} \times \omega_j^o \qquad \forall j \in \mathcal{O}, j \notin \mathcal{O}^{out} \quad (6)$$

A computational resource is multi-tenant and hosts one or more operators. Operators within the same host communicate directly whereas inter-node communication occurs via a communication service as depicted in Fig. 2. The employed queueing system is representative of the communication and processing services performed by DSPEs. Frameworks often run operators in containers, and the communication between them is via message queues or brokers [28]. After an operator processes a message, it is pushed to the queue(s) of the downstream operator(s). The Dispatching Service and Data Transfer Service provide functions analogous to a Stream Manager in Apache Heron or a Supervisor in Apache Storm. These services were decoupled from the local manager in our case because of the geo-distributed nature of cloud-edge infrastructure. The Dispatching Service manages the communication between operators locally while the Data Transfer Service handles external communications. Operators in the same resource write directly to their downstream operator queues while in external communications message brokers guarantee the message delivery.

Events are handled in a First-Come, First-Served (FCFS) fashion both by operators and the communication service that serializes events to be sent to another host. This guarantees the time order of events; an important requirement in many data stream processing applications. Both operators and the communication service follow a widely employed M/M/1 Queueing Theory model [29] for their queues which allows for estimating the waiting and service times for computation and communication. The used queue model is capable of capturing randomness in arrival and service times. As in state-of-the-art solutions [7]: (i) Event arrival follows a time-homogeneous Poisson process with a constant rate – without event bursts as sensors collect data periodically; and (ii) the service rate follows an exponential distribution with constant mean time – homogeneous processing demand for computing each message. Moreover, a stateful operator can have an impact on the computation time as it waits until it receives a number of events before considering the window complete (ws_i^o). The computation or service time $stime_{\langle i,k\rangle}$ in seconds of an operator i placed on resource k is hence given by:

$$stime_{\langle i,k\rangle} = \frac{1}{\mu_{\langle i,k\rangle} - \lambda_i^{in}} + \frac{ws_i^o}{\lambda_i^{in}} \quad (7)$$

The communication time $ctime_{\langle i,k\rangle\langle j,l\rangle}$ in seconds for operator i placed on a resource k to send an event to operator j on a resource l is:

$$ctime_{\langle i,k\rangle\langle j,l\rangle} = \frac{1}{\left(\frac{bdw_{k\leftrightarrow l}}{\varsigma_i^{out}}\right) - \lambda_j^{in}} + lat_{k\leftrightarrow l} \quad (8)$$

3.3 Infrastructure and Application Constraints

Edge resources can be limited in terms of memory, computing, and communication capabilities. An operator can only be placed on a computing resource if it meets all the operator's processing and data transfer requirements:

$$\lambda_i^{in} < \mu_{\langle i,k\rangle} \quad \forall i \in \mathcal{O}, \forall k \in \mathcal{R} | \mathbb{1}_{\langle i,k\rangle} = 1 \quad (9)$$

$$\lambda_i^{out} < \left(\frac{bdw_{k\leftrightarrow l}}{\varsigma_i^{out}}\right) \quad \forall i \in \mathcal{O}, \forall k \leftrightarrow l \in \mathcal{L} | \mathbb{1}_{\langle i,k\rangle} = 1 \quad (10)$$

The CPU and memory requirements of operators on each host are ensured by constraints 11 and 12:

$$\sum_{i \in \mathcal{O}} \mathbb{1}_{\langle i,k \rangle} \times \lambda_i^{in} \times cpu_i^o \leq cpu_k^r \qquad \forall k \in \mathcal{R} \tag{11}$$

$$\sum_{i \in \mathcal{O}} \mathbb{1}_{\langle i,k \rangle} \times (mem_i^o + ws_i^o \times \varsigma_i^{in}) \leq mem_k^r \quad \forall k \in \mathcal{R} \tag{12}$$

Data transferring requirements of streams placed on links are guaranteed by:

$$\sum_{\substack{e_{i \to j} \in \mathcal{E} \\ k \leftrightarrow l \in \mathcal{L}}} \mathbb{1}_{\langle e_{i \to j}, k \leftrightarrow l \rangle} \times \varsigma_i^{out} \leq bwd_{k \leftrightarrow l} \qquad \forall k \leftrightarrow l \in \mathcal{L} \tag{13}$$

Constraints 14 and 15 ensure that an operator is not placed on more than one resource and that a stream is not placed on more than a network link respectively:

$$\sum_{k \in \mathcal{R}} \mathbb{1}_{\langle i,k \rangle} = 1 \qquad \forall i \in \mathcal{O} \tag{14}$$

$$\sum_{k \leftrightarrow l \in \mathcal{L}} \mathbb{1}_{\langle e_{i \to j}, k \leftrightarrow l \rangle} = 1 \qquad \forall e_{i \to j} \in \mathcal{E} \tag{15}$$

3.4 End-to-End Latency

As DSP applications must handle incoming data events under short delays, the goal of the operator placement is to minimize the application end-to-end latency.

A *path* in a DSP application graph is a sequence of operators from a source to a sink. A path p_i of length n is a sequence of n operators and $n-1$ streams, starting at a source and ending at a sink:

$$p_i = o_0, o_1, \ldots, o_k, o_{k+1}, \ldots, o_{n-1}, o_n \tag{16}$$

where o_0 is a source and o_n is a sink. The set of all possible paths in the application graph is denoted by $\mathcal{P}$. The end-to-end latency of a path is the sum of the computation time of all operators along the path and the communication time required to stream events on the path. More formally, the end-to-end latency of path p_i, denoted by L_{p_i}, is:

$$\begin{aligned} L_{p_i} = & \sum_{j \in p_i, k \in \mathcal{R}} \mathbb{1}_{\langle j,k \rangle} \times stime_{\langle j,k \rangle} \\ & + \sum_{l \in \mathcal{R}} \mathbb{1}_{\langle e_{j \to j+1}, k \leftrightarrow l \rangle} \times ctime_{\langle j,k \rangle \langle j+1,l \rangle} \end{aligned} \tag{17}$$

The aggregate end-to-end latency AL is therefore:

$$AL = \sum_{p_i \in \mathcal{P}} L_{p_i} \tag{18}$$

As DSP applications are generally latency-sensitive, this paper provides solutions that minimize the *aggregate end-to-end latency*. In other words, find the mapping that minimizes AL (*i.e.*, $\min AL$) and respects the resource and network constraints.

4 Operator Placement Strategies

This section details the strategies proposed for placing DSP applications onto cloud-edge infrastructure while minimizing the aggregate end-to-end latency.

4.1 System Overview

This work considers a scenario where a user submits an application to the DSPE (*e.g.*, NebulaStream [30]), and an orchestrator running in the cloud computes the initial operator placement. The orchestrator maintains the network topology information and profiles the application to determine its operators' requirements and communication patterns. With the application and network topology information at hand, the orchestrator can employ one of the strategies described later to compute the placement. The applications are split according to the proposed strategies. Before applying a strategy, however, the orchestrator needs to create a deployment sequence, which is essentially a list of ordered operators, and the list of resources that can host the operators. The next section describes how the deployment sequence is computed, which is then used by all proposed strategies to evaluate operators in a sequential manner.

4.2 Deployment Sequence

The problem of determining the operator placement has been shown to be NP-hard [31]. In our strategies, we adopt an incremental and scalable approach for assigning operators to resources by evaluating a deployment sequence, built so that upstream operators are assessed before downstream operators. When estimating the requirements of a given operator, the system must guarantee that the requirements of its upstream operator have already been estimated. In this way, the placement strategy has all the information about the upstream operators such as an operator's output data rate which is needed when estimating the AL of a downstream operator. That is, the strategies require the output data rate from previous operators (*i.e.*, upstream operators) to estimate the requirements of a given downstream operator. After assigning all operators of the deployment sequence, the strategies can efficiently find an operator placement by minimizing AL of a given application dataflow.

Algorithm 1 shows the method for sorting operators and creating a deployment sequence. The method initializes the deployment sequence and a queue (line 1). As data sources do not have upstream operators, they are added to the deployment sequence along with their downstream operators, obtained via the function $GetDS$ (lines 2-6). Then for each operator in the queue, the algorithm checks whether its upstream operators are in the deployment sequence (lines 7-15). If all upstream operators are in the sequence, the operator is moved from the queue to the deployment sequence and its downstream operators are pushed to the queue (lines 9-13). Otherwise, the operator is moved from the beginning to the end of the queue (line 15). The evaluation of upstream operators completes when the queue is empty. As Algorithm 1 essentially performs a breadth-first traversal of the application dataflow, its complexity is $O(|\mathcal{O}| + |\mathcal{E}|)$, where $\mathcal{O}$ is set of application operators and $\mathcal{E}$ is the set of streams in the dataflow.

4.3 Operator Placement Strategies

This section introduces three operator placement strategies that use the deployment sequence.

Algorithm 1 to build the deployment sequence.

```
1:  ds ← { }, queue.clear()
2:  for src in O^src do
3:      ds.append(src)
4:      for downs in GETDS(src) do
5:          if downs not in queue then
6:              queue.push(downs)
7:  while |queue| > 0 do
8:      ope ← queue.pop()
9:      if all upstream operators of ope are in ds then
10:         for downs in GETDS(ope) do
11:             if downs not in queue then
12:                 queue.push(downs)
13:         ds.append(ope)
14:     else
15:         queue.push(ope)
16: return ds
```

4.3.1 Aggregate End-to-End Latency Strategy (AELS)

AELS is a greedy strategy that places operators incrementally by evaluating the AL while respecting the resource constraints presented in Section 3. AELS estimates the AL

Algorithm 2 to compute the end-to-end latency.

```
1:  function ESTIMATEAL(N = (R, L), G = (O, S), o)
2:      rt ← {}
3:      us ← placements of operators upstream to o
4:      for r ∈ R do
5:          comm ← 0
6:          for plcmt ∈ us do
7:              if GETHOST(plcmt) ≠ r then
8:                  comm ← comm + ctime_⟨plcmt⟩⟨o,r⟩
9:          if MEETCONSTRAINTS(o, r) then
10:             rt.append(⟨r, stime_⟨o,r⟩ + comm⟩)
11:     return rt
```

for each operator in the deployment sequence by considering the already placed upstream operators, resource capabilities, and operator requirements. Algorithm 2 shows this process, where the algorithm first obtains all the operators that are upstream to operator o for estimating the communication overhead from them to operator o itself (line 3). The placement of operator o is then evaluated on all resources (lines 4-10). During the evaluation, the communication overhead is computed according to the placement of upstream operators (lines 5-8). If a resource can provide the CPU, memory and bandwidth required by operator o, the resource and its end-to-end latency are added to a set of available resources (lines 9-10). After that, the resources are sorted by their end-to-end latencies. The resource with the smallest end-to-end latency is picked, and the resource's residual CPU and memory capabilities are updated. If the application graph were complete, the complexity of executing Algorithm 2 for all operators in the sequence would be $O(|\mathcal{O}| \times |\mathcal{R}| + |\mathcal{O}|^2)$, where $\mathcal{O}$ is the set of application operators and $\mathcal{R}$ is the set of infrastructure resources. As DSP graphs are often more sparse, the cost of computing the communication latency from upstream streams is negligible

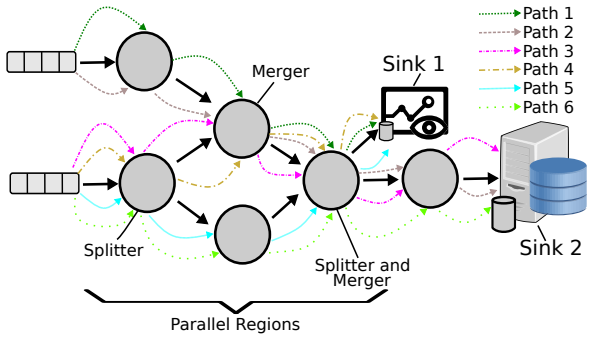


Fig. 3. Example of application dataflow with several operator patterns, resulting in multiple application paths, each with its own performance requirements. Each arrow type corresponds to a unique path.

and the complexity can be considered $O(|\mathcal{O}| \times |\mathcal{R}|)$. As described in the next section, DSP applications often comprise parallel regions whose operators AL could be evaluated in parallel. Also, the next sections present strategies that further reduce this complexity.

4.3.2 AELS with Region Patterns (AELS+RP)

Aggregate End-to-End Latency Strategy with Region Patterns (AELS+RP) is a strategy that handles complex dataflows that contain multiple paths from sources to sinks. As shown in Figure 3, a DSP dataflow can have patterns such as: (i) splitters, where messages are replicated to multiple downstream operators or scheduled to downstream operators in a round-robin fashion using message key hashes or other criteria [32]; (ii) parallel regions that perform the same operations over different sets of messages or where each individual region treats copies of the incoming messages; and (iii) mergers, which merge the outputs of parallel regions. These patterns result in multiple application paths, each with its own performance requirements. For instance, messages emitted to *sink 1* require shorter AL than those sent to be stored in databases, *i.e.*, *sink 2*. AELS+RP considers each application path requirements and gives priority to messages according to their destination. This information is used to build regions and assist in placing operators across cloud and edge resources. AELS+RP hence explores splitters and the sinks' placement (cloud or edge). According to the destination of the output of an operator, this strategy classifies the regions in:

- *cloud* when an operator is immediately upstream to sinks placed on the cloud; and
- *edge* when an operator shares paths with sinks located at the edge.

Based on the regions, the operators are classified and allocated as depicted in Fig. 4. Since an *edge region* has a higher priority than a *cloud region*, AELS+RP exploits resources on edge sites for improving the AL. If an operator exceeds the capabilities of edge resources, then it is allocated to one of the clouds. AELS+RP aims to allocate operators across edge and cloud meeting the AL only for operators on the edge region, in contrast to the AELS strategy that evaluates the end-to-end latency rate for all operators.

Similar to AELS, when evaluating the AL for an operator in an edge region, the resource with the shortest AL is

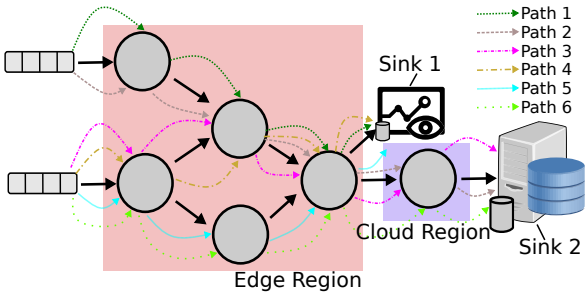


Fig. 4. The application is divided into regions according to the destination of messages. Each square box corresponds to a given priority for the operator placement. Operators on the edge region (red square box) have priority to be placed on resources at the edge sites while cloud region (grey square box) are directed to the cloud provider closest to the data sink.

picked. Operators on the edge region will be evaluated considering only resources on edge sites, unless their requirements are not met. The cloud will host operators in the cloud region and those of the edge region whose requirements cannot be satisfied by resources on edge sites.

4.3.3 AELS+RP and Latency Awareness (AELS+RP+LA)

AELS+RP improves the scalability of policies introduced in our previous work [13]. AELS+RP reduces the search space by picking a selected number of operators whose placement is evaluated considering only edge resources instead of the overall infrastructure. Further reducing the number of evaluations of operator assignments is key to finding operator placements in feasible time as current and future IoT scenarios can contain many thousands of resources [8]. The AELS+RP+LA improves the scalability of AELS+RP by applying a pruning technique to reduce the number of resources and estimate the AL.

As previous work has shown that network latency is often the largest overhead in a geo-distributed infrastructure [26], AELS+RP+LA explores it to reduce the number of resources. When computing the incremental operator placement from the deployment sequence, each operator has its AL computed on a limited number of resources. These resources[1] are added to a list according to the following schemes:

- **In-situ**: gets the best resources to place a given operator to avoid communication across edge sites. First, it identifies the upstream operators of a downstream operator. Second, each upstream operator placed on edge has the resources with the smallest network latency in the same edge site added to the considered resource list.
- **In-transit**: obtains resources to reduce the overhead of communication across edge sites. The approach also evaluates the location of each upstream operator. For each one, the method selects a resource in an edge site (if the upstream operator is placed on the edge, then the obtained edge site must be different from the current one) and a cloud resource according to their network latency. Then resources are included to the considered resource list.
- **Data sink locations**: adds to the considered resource list the resources where the destination data sinks are placed. For each different data sink location, the method includes the resource with the smallest network latency.

Once the resource list is compiled, it is submitted to the AELS+RP strategy. Instead of examining all edge sites and clouds, the search space covers only resources that can impact the AL positively.

1. Only resources that support the memory, CPU and bandwidth requirements of the operator are considered.

5 Evaluation

This section first describes the experimental setup and performance metrics and then discusses obtained results.

5.1 Experimental setup

We built a framework atop OMNET++ [33] to model and simulate the large-scale DSP scenarios envisioned in this work[2]. Our previous work [22] showed that under smaller-scale scenarios, simulation results closely match those obtained in real-life settings. A computational resource is an entity with CPU, memory and bandwidth capabilities whereas operators comprise waiting queues and transformation operations posing demands in terms of CPU, memory and bandwidth.

The edge resources are modeled as Raspberry PIs 2 (RPi2) (*i.e.*, 4.74 MIPS at 1 GHz and 1 GB of RAM) and Raspberry PIs 3 (RPi3) (*i.e.*, 5.02 MIPS at 1.2 GHz and 1 GB of RAM). The cloud resources are modeled as AMD RYZEN 7 1800x (*i.e.*, 304.51 MIPS [34] at 3.6 GHz and 1 TB of memory). Half the edge resources are RPi2 and the other half are RPi3. We evaluate multiple network configurations (or topologies) by varying the number of edge sites, clouds and IoT resources in each site from the following values: 10, 100 and 500. The configurations are summarized in Table 3. The network topologies are classified as:

- **Regular**: with less than 10,000 resources.
- **Large**: between 10,000 and 99,999 resources.
- **Extra-large**: has 100,000 or more resources.

The network topologies follow patterns obtained from the work by Hu *et al.* [26]:

- A gateway interfaces each edge site's LAN and the external WAN (the Internet).
- The LAN latency is uniformly distributed between 0.015 and 0.8 ms with a bandwidth of 100 Mbps.
- The WAN latency is drawn uniformly between 65 and 85 ms, and the bandwidth is 1 Gbps.

Aiming to capture a diverse range of IoT and monitoring applications where sensors/actuators generate a variety of events, 13 application graphs with single and multiple data paths are considered divided into three categories as described later. The 13 graphs were crafted using a Python

2. https://github.com/aveith/simulator-stream-processing

TABLE 3
Network topologies and their classifications.

Topology	Number of Resources			
	Cloud	Edge Site	IoTs	Total
Regular	10	10	10	110
	100	10	10	200
	500	10	10	600
	10	10	100	1,010
	10	100	10	1,010
	100	10	100	1,100
	100	100	10	1,100
	500	10	100	1,500
	500	100	10	1,500
	10	10	500	5,010
	10	500	10	5,010
	100	10	500	5,100
	100	500	10	5,100
	500	10	500	5,500
	500	500	10	5,500
Large	10	100	100	10,010
	100	100	100	10,100
	500	100	100	10,500
	10	100	500	50,010
	10	500	100	50,010
	100	100	500	50,100
	100	500	100	50,100
	500	100	500	50,500
	500	500	100	50,500
Extra-large	10	500	500	250,010
	100	500	500	250,100
	500	500	500	250,500

library[3] and their orders are based on the size of Real-time IoT Benchmark (RIoTBench) topologies [35], which are representative of today's DSP applications. The number of stateful operators corresponds to 20% of the whole application, also based on RIoTBench topologies. The sources and sinks are placed on edge sites except for the sink on the critical path, which is hosted on the cloud. This is the typical behavior of IoT and smart home applications that collect data from sensors located on the edge of the Internet and have to provide response to nearby actuators, whereas part of the processing is performed in the cloud [36], [37]. The experiments consider 10 different configurations for each of the 13 application graphs, where operator parameters and the location of sinks and sources are uniformly drawn from the value ranges shown in Table 4, hence resulting in a total of 130 application graph settings divided into:[4].

- **Medium**: 5 applications with up to 9 operators.
- **Large**: 7 applications with 25 operators.
- **Extra-large**: 1 application with 50 operators.

The size of *large* and *extra-large* graphs is much larger than that of existing data stream processing applications, and by having several sources and splitters, the resulting paths can be viewed as multiple applications with several sense-process/actuation loops. This captures the behavior wherein the execution of an application (sense-process loop) triggers the execution of another application.

An operator placement computed by a strategy is executed for 60 seconds and compared against a traditional

3. https://gist.github.com/bwbaugh/4602818

4. The operator parameters are based on the work by Murtaza *et al.* [38] and Shukla *et al.* [35] who analyzed real-world applications.

TABLE 4
Operator parameters in the application graphs.

Parameter	Value	Unit
cpu	1,000-10,000	Instructions per second
Data compression rate	10-100	%
mem	100-7,500	bytes
Input event size	100-2,500	bytes
Selectivity	10-100	%
Input event rate	1,000-10,000	Messages per second
ws (window size)	1-100	Number of messages

deployment approach (Cloud-only) and TCEP [3] from the state-of-the-art that performs cloud-edge placement. *Cloud-only* deploys all operators on the cloud. TCEP iterates a vector containing the application operators, and for each operator, the algorithm ranks the computational resources by CPU, gets the host of the middle of the rank, and evaluates CPU, memory, and bandwidth constraints to obtain the operator placement. If there exists any constraint, the operator is assigned to the cloud. TCEP is chosen for comparison because it is the closest solution to ours regarding the features examined in Section 2. Also, it has fewer scalability issues and could potentially address some of the large-scale settings considered here.

Performance Metrics:

- **Resolution time**: the time in seconds to obtain a solution according to the used strategy (*i.e.*, TCEP, cloud-only, AELS, AELS+RP, and AELS+RP+LA).
- **Execution time violations**: the number of unsuccessful runs where the operator placement strategy exceeded the resolution time limit of 600 seconds[5].
- **Aggregate end-to-end latency**: the end-to-end latency in seconds from the time events are generated to the time they are processed by the sinks, considering application settings without time violations.

5.2 Performance Evaluation Results

This section presents the performance evaluation results for regular, large and extra-large network topologies.

5.2.1 Regular Network Topology

Table 5 summarizes the execution time violations for 1,950 experiments conducted with regular network topologies. The results are organized according to the application sizes. For *AELS*, *AELS+RP* and *Cloud-only*, the number of violations increase with the number of operators. This is due to the explosion of combinatorial space when considering resources in each strategy. These strategies do not have a significant reduction in the number of resources and operators because they employ weak methods for reducing such numbers. The strategies must evaluate each operator considering a large number of resources.

AELS has the highest number of violations because it checks at least once the end-to-end latency of an operator

5. Due to the dynamic nature of DSP applications and edge computing, the operators needs and resource availability can change in very short periods of time. Hence we adopt 10 minutes as a worst case resolution time within which a scheduler needs to compute a feasible operator placement.

for each resource in the network topology. In contrast, *AELS+RP* and *Cloud-only* apply a simple pruning approach, slightly decreasing the number of violations. *Cloud-only* covers only cloud resources in its search space, while *AELS+RP* reduces both the number of resources and operators to be evaluated. *AELS+RP* forces the usage of IoT resources on the edge and places operators in the cloud when no edge resource matching the operator requirements is found. *Cloud-only* achieves fewer violations compared to *AELS+RP* because its search space is at least 10 times smaller when comparing the maximum number of resources considered in the experiments. This demonstrates a strong relationship between the number of considered resources and the number of violations, which results in small number of violations under *TCEP* and *AELS+RP+LA*. *AELS+RP+LA* executes a quick search to the resources organizing them by their bandwidth capacities and then selects fewer resources according to three selection approaches. *TCEP* iterates a vector containing the application operators, gets the middle host of the available resources and evaluates CPU, memory, and bandwidth constraints to obtain the operator placement. Hence, the strategy has a Best Fit method, which allows it to compute solutions in a feasible time.

TABLE 5
Percentage of execution time violations for application sizes (medium, large, and extra-large) in regular network topologies.

Solution	Medium(%)	Large(%)	Extra-large(%)
AELS	52	65	71
AELS+RP	48	59	70
AELS+RP+LA	0	0	0
Cloud-only	40	41	46
TCEP	0	0	0

Fig. 5 presents the results for the resolution time for each application size. *AELS+RP+LA* outperforms *AELS*, *AELS+RP* and *Cloud-only* in over 77% when comparing the mean resolution time as depicted in Fig. 5. The resolution time reflects the number of resources evaluated for each operator. *AELS* has the worst resolution time because it assesses all cloud and IoT resources while *AELS+RP* and *Cloud-only* reduce the resolution time by considering the set of IoT resources or the set of cloud resources, respectively. *TCEP* has longer resolution times of over 17% for extra-large applications, ≈48% in large applications, and ≈83% in medium applications when compared to *AELS+RP+LA*. The gradual performance improvement of *TCEP* is due to the way operators and resources are sorted in their respective vectors. At the same time, *AELS+RP+LA* employs a strategy to elect resources to be considered at each operator iteration, resulting in a progressive expansion of the resolution time when rising the number of operators.

AELS+RP+LA enhances *AELS+RP* by reducing the number of resources according to their network latencies, which poses the highest overhead when dealing with geographically distributed infrastructure. This enables *AELS+RP+LA* to have an aggregate end-to-end latency similar (≈3% higher) to *AELS+RP* as depicted in Fig. 6. Similar to *AELS+RP*, *AELS* computes each operator aggregate end-to-end latency incrementally for each resource, the incremental method with this high search space brings a performance loss of ≈3% compared to *AELS+RP+LA*. AELS+RP+LA outperforms *Cloud-only* and *TCEP* by over 26% and 50% respectively. *Cloud-only* has performance loss in application paths where data sinks are placed on the edge, requiring messages to traverse network links with high latencies at least twice. Meanwhile, *TCEP* achieves the worst aggregate end-to-end latency because it comprises general vectors for resources and operators, regardless of the communication overhead.

5.2.2 Large Network Topology

Table 6 presents the results of 1,170 experiments where network topologies have a number of resources between 10k and 100k. As most state-of-the-art solutions rely on regular network topologies without numerous edge resources or federated clouds, they present a considerable number of violations. *Cloud-only* fails mainly in experiments where the number of cloud resources is equal to 500 because it evaluates all cloud resources when assessing an operator. Similarly, *TCEP* increases its number of violations because the number of considered resources is greater than or equal to 50,000, which results in a high overhead for sorting the resource vector at each operator evaluation. AELS+RP+LA has no violations because it smartly selects strategic resources when estimating the communication and processing time of each operator.

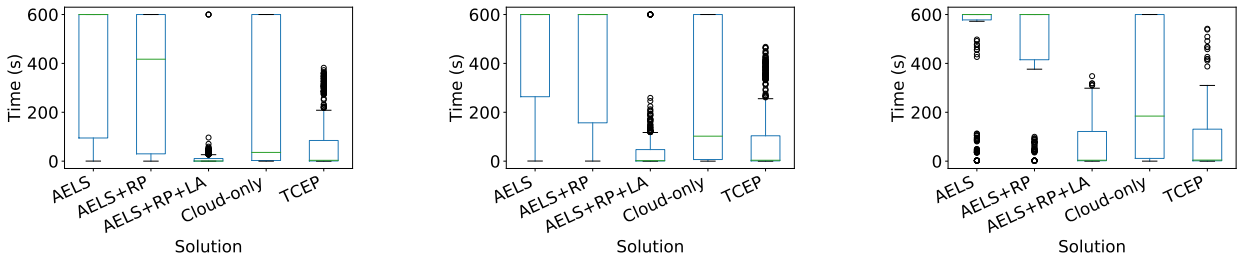
TABLE 6
Percentage of execution time violations for application sizes (medium, large, and extra-large) in large network topologies.

Solution	Medium(%)	Large(%)	Extra-large(%)
AELS	100	100	100
AELS+RP	94	100	100
AELS+RP+LA	0	0	0
Cloud-only	45	60	67
TCEP	44	44	44

AELS and *AELS+RP* often exceed the time limit of 600 seconds and hence seldom compute operator placements, as depicted in Fig. 7. Although *Cloud-only* and *TCEP* find solutions, they require at least 65% more time and yield ≈44% higher aggregate latency when compared to AELS+RP+LA, as presented in Fig. 8. Moreover, AELS+RP+LA increases the resolution time by ≈37% compared to the regular network topology, but it keeps the aggregate end-to-end latency nearly stable, with a slight loss ≈2%. The growth in resolution time is caused by the overhead in selecting resources considered for operator placement; expected as the strategy covers numerous resources. Nevertheless, our approach effectively maintains the aggregate end-to-end latency by picking resources according to their network latency, which reflects the overhead of message processing.

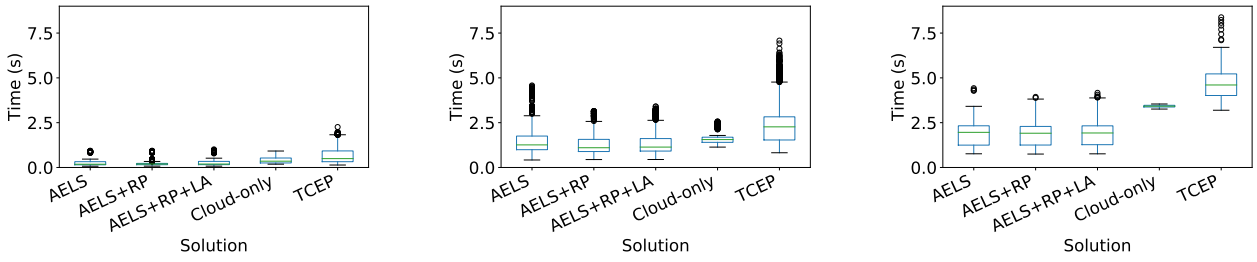
5.2.3 Extra-large Network Topology

Here we consider future large-scale scenarios where IoT and edge resources become very pervasive [8]. This set comprises 390 experiments for network topologies with more than 100k resources. These experiments show that few strategies can produce valid placements for extra-large topologies within the allotted execution time limit as depicted in Table 7. In most cases, *Cloud-only* and



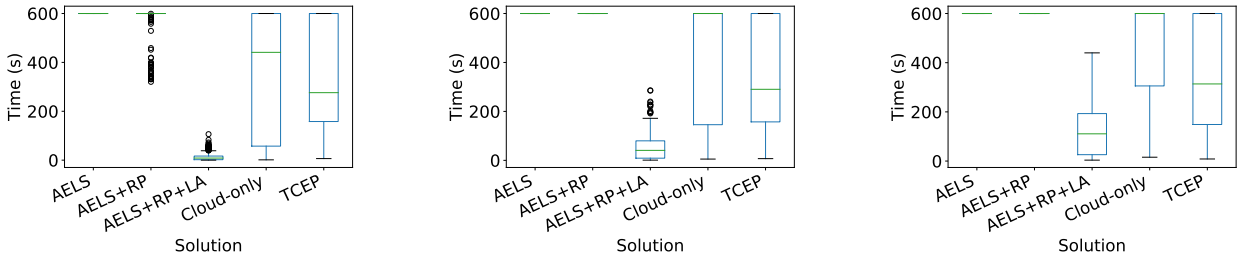
(a) Medium Application Sizes (b) Large Application Sizes (c) Extra-large Application Sizes

Fig. 5. Resolution time for network topologies with less than 10k resources.



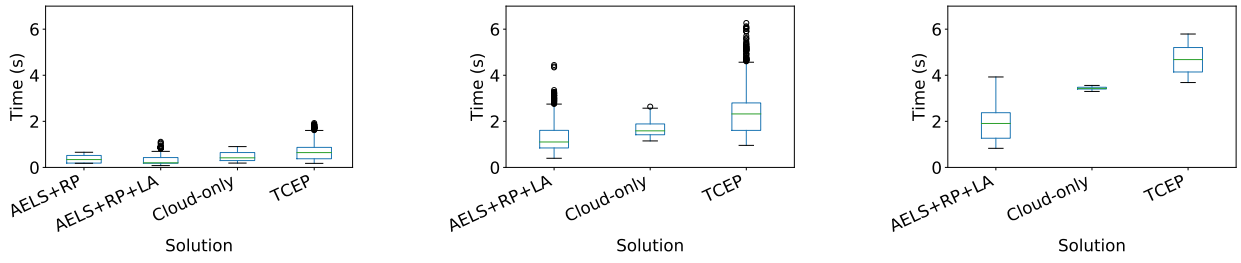
(a) Medium Application Sizes (b) Large Application Sizes (c) Extra-large Application Sizes

Fig. 6. Aggregate end-to-end latency for network topologies with less than 10k resources.



(a) Medium Application Sizes (b) Large Application Sizes (c) Extra-large Application Sizes

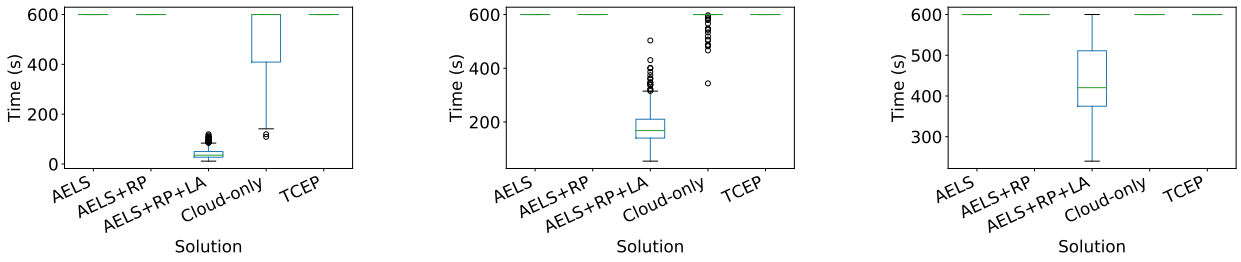
Fig. 7. Resolution time for network topologies greater than 10k and smaller than 100k resources. AELS and AELS+RP often exceed the time limit of 600 seconds, and hence compute very few placement solutions.



(a) Medium Application Sizes (b) Large Application Sizes (c) Extra-large Application Sizes

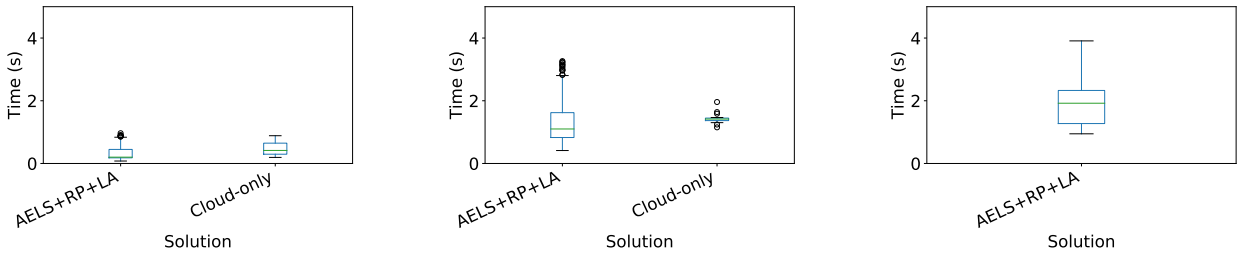
Fig. 8. Aggregate end-to-end latency for network topologies greater than 10k and smaller than 100k resources. AELS cannot provide any operator placement because it exceeds the time limit of 600 seconds. AELS+RP is able to produce solutions only for medium application sizes.

AELS+RP+LA can produce operator placements. However, *Cloud-only* has higher time violations than *AELS+RP+LA*. Our method only fails in some cases for extra-large applications where the number of resources is equal to 250,500 (worst case) because some application configurations require more time to estimate the AL. This time demand



(a) Medium Application Sizes (b) Large Application Sizes (c) Extra-large Application Sizes

Fig. 9. Results of resolution time for network topologies greater than 100k resources.



(a) Medium Application Sizes (b) Large Application Sizes (c) Extra-large Application Sizes

Fig. 10. Aggregate end-to-end latency for network topologies greater than 100k resources. In this set of experiments, the AELS, AELS+RPs and TCEP approaches cannot provide any operator placement to medium and large application sizes because they exceed the resolution time of 600 seconds. In contrast, Cloud-only is not able to produce operator placements only for extra-large application sizes.

is caused by the heavy operator requirements and time to compute the expected latency for the evaluated resources.

TABLE 7
Percentage of execution time violations for application sizes (medium, large, and extra-large) in extra-large network topologies.

Solution	Medium(%)	Large(%)	Extra-large(%)
AELS	100	100	100
AELS+RP	100	100	100
AELS+RP+LA	0	0	7
Cloud-only	67	94	100
TCEP	100	100	100

Moreover, *AELS+RP+LA* increases the number of resources by 5 times when compared to large network topologies and the resolution time by ≈70%, as depicted in Fig. 9. However, these experiments demonstrate that *AELS+RP+LA* increases the aggregate end-to-end latency by less than 1%, as presented in Fig. 10, which shows its effectiveness to stabilize the *AL* regardless of the size of the network. Since *AELS+RP+LA* is based on the network latency, the results show how impactful the network latency is when handling time-sensitive applications.

6 Conclusions and Future Work

This work described three strategies to minimize the end-to-end latency of DSP applications by splitting the application graph and distributing operators across cloud and edge computing resources. The strategies leverage application behaviors for identifying regions to which operators belong and how certain resources can impact the operator placement. We started with Aggregate End-to-End Latency Strategy (AELS), which estimates the application end-to-end latency of each operator on all available resources. Aggregate End-to-End Latency Strategy with Region Patterns (AELS+RP) splits the dataflow graph using region patterns and estimates the end-to-end latency only for operators that are candidates to be deployed on edge sites. Aggregate End-to-End Latency Strategy with Region Patterns and Latency Awareness (AELS+RP+LA) further reduces the number of evaluated resources by considering the resource network latencies when assessing the regions identified by AELS+RP.

Performance evaluation results show that the proposed strategies can achieve at least 30% better end-to-end latency than cloud-only and a state-of-the-art solution when applications have multiple forks, parallel regions and joins. The results also demonstrate that Aggregate End-to-End Latency Strategy with Region Patterns and Latency Awareness (AELS+RP+LA) is effective in maintaining the end-to-end latency regardless of the size of the network, whereas cloud-only and the state-of-the-art solution cannot produce an operator placement in feasible time. In future work, we intend to implement our strategies in a real DSPE.

References

[1] B. Heintz, A. Chandra, and R. K. Sitaraman, "Optimizing timeliness and cost in geo-distributed streaming analytics," *IEEE Transactions on Cloud Computing*, vol. 8, no. 1, pp. 232–245, 2020.

[2] H. P. Sajjad, K. Danniswara, A. Al-Shishtawy, and V. Vlassov, "Spanedge: Towards unifying stream processing over central and near-the-edge data centers," in *2016 IEEE/ACM Symp. on Edge Comp.*, Oct 2016, pp. 168–178.

[3] M. Taneja and A. Davy, "Resource aware placement of IoT application modules in Fog-Cloud Computing Paradigm," in *IFIP/IEEE Symp. on Integrated Net. and Service Management (IM)*, May 2017, pp. 1222–1228.

[4] C. Hochreiner, M. Vogler, P. Waibel, and S. Dustdar, "VISP: An ecosystem for elastic data stream processing for the internet of things," in *20th IEEE Int. Enterprise Distributed Object Computing Conf.*, Sept 2016, pp. 1–11.

[5] A. J. Fahs and G. Pierre, "Tail-latency-aware fog application replica placement," in *Service-Oriented Computing*, E. Kafeza, B. Benatallah, F. Martinelli, H. Hacid, A. Bouguettaya, and H. Motahari, Eds. Cham: Springer International Publishing, 2020, pp. 508–524.

[6] Q. Fan and N. Ansari, "Application aware workload allocation for edge computing-based IoT," *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 2146–2153, 2018.

[7] V. Cardellini, F. Lo Presti, M. Nardelli, and G. Russo Russo, "Optimal operator deployment and replication for elastic distributed data stream processing," *Concurrency and Computation: Practice and Experience*, vol. 30, no. 9, p. e4334, 2018.

[8] J. Gedeon, M. Stein, J. Krisztinkovics, P. Felka, K. Keller, C. Meurisch, L. Wang, and M. Mhlhuser, "From cell towers to smart street lamps: Placing cloudlets on existing urban infrastructures," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, Oct 2018, pp. 187–202.

[9] "Humble Lamppost," https://eu-smartcities.eu/initiatives/78/description, 2018, online.

[10] "Link NYC," https://www.link.nyc/, 2019, online.

[11] "Smight," https://smight.com/, 2018, online in German.

[12] "EdgeMicro," https://www.edgemicro.com/, 2019, online.

[13] A. da Silva Veith, M. D. de Assunção, and L. Lefèvre, "Latency-aware placement of data stream analytics on edge computing," in *Service-Oriented Computing*, C. Pahl, M. Vukovic, J. Yin, and Q. Yu, Eds. Cham: Springer International Publishing, 2018, pp. 215–229.

[14] H. Röger and R. Mayer, "A comprehensive survey on parallelization and elasticity in stream processing," *ACM Comput. Surv.*, vol. 52, no. 2, Apr. 2019.

[15] R. Mahmud, K. Ramamohanarao, and R. Buyya, "Edge affinity-based management of applications in fog computing environments," in *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*, ser. UCC19. New York, USA: Association for Computing Machinery, 2019, p. 6170.

[16] T. Elgamal, A. Sandur, P. Nguyen, K. Nahrstedt, and G. Agha, "DROPLET: Distributed operator placement for IoT applications spanning edge and cloud resources," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, 2018, pp. 1–8.

[17] L. Gu, D. Zeng, S. Guo, Y. Xiang, and J. Hu, "A general communication cost optimization framework for big data stream processing in geo-distributed data centers," *IEEE Transactions on Computers*, vol. 65, no. 1, pp. 19–29, Jan 2016.

[18] W. Chen, I. Paik, and Z. Li, "Cost-aware streaming workflow allocation on geo-distributed data centers," *IEEE Transactions on Computers*, vol. 66, no. 2, pp. 256–271, Feb 2017.

[19] C. Canali and R. Lancellotti, "GASP: genetic algorithms for service placement in fog computing systems," *Algorithms*, vol. 12, no. 10, p. 201, 2019.

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

[21] A. Kiani and N. Ansari, "Optimal code partitioning over time and hierarchical cloudlets," *IEEE Communications Letters*, vol. 22, no. 1, pp. 181–184, 2018.

[22] E. Gibert Renart, A. Da Silva Veith, D. Balouek-Thomert, M. D. De Assuno, L. Lefèvre, and M. Parashar, "Distributed operator placement for IoT data analytics across edge and cloud resources," in *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, May 2019, pp. 459–468.

[23] "Amazon EC2," https://aws.amazon.com/ec2/.

[24] "Microsoft Azure," https://azure.microsoft.com/en-ca/.

[25] "Google Cloud," https://cloud.google.com/.

[26] W. Hu, Y. Gao, K. Ha, J. Wang, B. Amos, Z. Chen, P. Pillai, and M. Satyanarayanan, "Quantifying the impact of edge computing on mobile applications," in *7th ACM SIGOPS Asia-Pacific Wksp on Systems*, ser. APSys '16. New York, USA: ACM, 2016, pp. 5:1–5:8.

[27] H. Arkian, G. Pierre, J. Tordsson, and E. Elmroth, "An experiment-driven performance model of stream processing operators in Fog computing environments," in *ACM/SIGAPP Symp. On Applied Computing (SAC 2020)*, Brno, Czech Republic, Mar. 2020.

[28] M. Sarathchandra, C. Karandana, W. Heenatigala, M. Dayarathna, and S. Jayasena, "Resource aware scheduler for distributed stream processing in cloud native environments," *Concurrency and Computation: Practice and Experience*, vol. n/a, no. n/a, p. e6373. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.6373

[29] R. Ghosh, S. P. R. Komma, and Y. Simmhan, "Adaptive energy-aware scheduling of dynamic event analytics across edge and cloud resources," in *Proceedings of the 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, ser. CCGrid '18. Piscataway, USA: IEEE Press, 2018, pp. 72–82.

[30] S. Zeuch, E. T. Zacharatou, S. Zhang, X. Chatziliadis, A. Chaudhary, B. D. Monte, D. Giouroukis, P. M. Grulich, A. Ziehn, and V. Mark, "Nebulastream: Complex analytics beyond the cloud," *Open Journal of IoT (OJIOT)*, vol. 6, no. 1, pp. 66–81, 2020.

[31] A. Benoit, A. Dobrila, J.-M. Nicod, and L. Philippe, "Scheduling linear chain streaming applications on heterogeneous systems with failures," *Future Gener. Comput. Syst.*, vol. 29, no. 5, pp. 1140–1151, Jul. 2013.

[32] H. Chen, F. Zhang, and H. Jin, "Pstream: a popularity-aware differentiated distributed stream processing system," *IEEE Transactions on Computers*, pp. 1–1, 2020.

[33] "OMNeT++," https://omnetpp.org/, 2017, online.

[34] "BOINC Performance on AMD Ryzen," https://www.reddit.com/r/BOINC/comments/5xog5v/boinc_performance_on_amd_ryzen/, 2018, online.

[35] A. Shukla, S. Chaturvedi, and Y. Simmhan, "RIoTBench: An IoT benchmark for distributed stream processing systems," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 21, p. e4257, 2017.

[36] F. A. Kraemer, A. E. Braten, N. Tamkittikhun, and D. Palma, "Fog computing in healthcare – a review and discussion," *IEEE Access*, vol. 5, pp. 9206–9222, 2017.

[37] N. Hassan, S. Gillani, E. Ahmed, I. Yaqoob, and M. Imran, "The role of edge computing in internet of things," *IEEE Communications Magazine*, vol. 56, no. 11, pp. 110–115, 2018.

[38] F. Murtaza, A. Akhunzada, S. Islam, J. Boudjadar, and R. Buyya, "Qos-aware service provisioning in fog computing," *Journal of Network and Computer Applications*, 04 2020.

Alexandre da Silva Veith is a postdoctoral researcher at the University of Toronto, Canada. He holds a Ph.D. in Computer Science from the Ecole Normale Superieure (ENS) of Lyon and the University of Lyon. His interests include reinforcement learning, IoT and data stream processing on cloud-edge infrastructure.

Marcos Dias de Assuncao is an associate professor at ETS Montreal, Canada. Before joining ETS, he was a former Researcher at Inria, France (2014–2020), and a former research scientist at IBM Research Brazil (2011–2014). He holds a Ph.D. in Computer Science and Software Engineering (2009) from The University of Melbourne, Australia.

Laurent Lefevre is a permanent researcher in computer science at Inria. He holds a Ph.D. in Computer Science from the Ecole Normale Superieure (ENS) of Lyon. His interests include energy efficiency, environmental impacts and performance of large-scale distributed computing and networking infrastructures, high performance computing, high performance networks protocols and services.