An Optimal Model for Optimizing the Placement and Parallelism of Data Stream Processing Applications on Cloud-Edge Computing

Felipe Rodrigo de Souza, Marcos Dias de Assunção, Eddy Caron
Univ. Lyon, EnsL, UCBL, CNRS, Inria, LIP
LYON Cedex 07, France
{felipe-rodrigo.de-souza, marcos.dias.de.assuncao, eddy.caron}@ens-lyon.fr

Alexandre da Silva Veith
University of Toronto
alexandre.veith@utoronto.ca

***Abstract*—The Internet of Things has enabled many application scenarios where a large number of connected devices generate unbounded streams of data, often processed by data stream processing frameworks deployed in the cloud. Edge computing enables offloading processing from the cloud and placing it close to where the data is generated, thereby reducing the time to process data events and deployment costs. However, edge resources are more computationally constrained than their cloud counterparts, raising two interrelated issues, namely deciding on the parallelism of processing tasks (a.k.a. operators) and their mapping onto available resources. In this work, we formulate the scenario of operator placement and parallelism as an optimal mixed-integer linear programming problem. The proposed model is termed as Cloud-Edge data Stream Placement (CESP). Experimental results using discrete-event simulation demonstrate that CESP can achieve an end-to-end latency at least $\simeq$ 80% and monetary costs at least $\simeq$ 30% better than traditional cloud deployment.**

***Index Terms*—Data Stream Processing, Operator Placement, Operator Parallelism, End-to-end Latency, Edge Computing.**

I. Introduction

Advances in Internet of Things (IoT) have led to a massive growth in the number of connected devices that generate ever-increasing data streams requiring timely processing. A Data Stream Processing (DSP) application is often structured as a directed graph whose *vertices* represent data sources, operators that execute a function over incoming data [1], and data sinks; and *edges* that define the data interdependencies between operators.

DSP applications are often deployed in the cloud to explore the large number of available resources and benefit from its pay-as-you-go business model. Moreover, the growth of IoT is creating scenarios where geo-distributed resources at the edge of the network act both as data sources and actuators or consumers of processed data. Streaming all this data to a cloud through the Internet, and sometimes back, takes time and quickly becomes costly [1]. Recent work has explored edge computing to address this problem by offloading DSP operators from the cloud and placing them closer to where data is often generated.

Cloud-edge computing is a computational paradigm that combines cloud and edge resources in a mix of public and private infrastructure, reducing the effects of the network in the application performance, where the edge is composed of IoT and Micro Datacenter (MD) resources. An inherent problem relies upon deciding how much and which parts of a DSP application should be offloaded from the cloud to resources elsewhere, a problem commonly known as *operator placement*. This problem, known to be NP-Hard [2], consists of finding a set of physical resources to host operators while respecting the application requirements. The search space can become very large as the number of resources and their heterogeneity increase.

Moreover, edge resources are often more constrained than those in the cloud. When offloading operators from the cloud, the DSP framework needs to adjust the degree of operators' parallelism through the creation of replicas to achieve a target throughput. The operator placement thus needs to address two interrelated issues, namely deciding on the number of instances for each operator and finding the set of resources to host the instances – while guaranteeing Quality of Service (QoS) metrics such as application throughput and end-to-end latency. Adding an extra level of complexity, the deployment of DSP applications in public infrastructures, such as cloud or MD, incurs monetary costs, which the decisions regarding parallelism, placement, and requirements for each parallel instance should consider.

This work introduces an optimal Mixed-Integer Linear Programming (MILP) model for determining the degree of parallelism and placement of DSP applications onto the cloud-edge infrastructure. We devise a solution for estimating the number of replicas, and the processing and bandwidth requirements of each operator to respect a given throughput and minimize the application end-to-end latency and deployment costs. Therefore, the contributions of this work are as follows:

- A MILP model called Cloud-Edge data Stream Placement (CESP) for the joint-optimization of operator parallelism and placement on edge and cloud computing to minimize the application end-to-end latency and deployment costs (section II).
- Experimental results demonstrating that CESP can improve the end-to-end latency by at least $\simeq$ 80% and

costs by at least $\simeq$ 30% compared to traditional cloud deployment (section III).

The rest of this paper is structured as follows. Section II introduces CESP for joint optimization of operator parallelism and placement. The experimental setup and performance results are presented in Section III. Related work is discussed in Section IV, whereas Section V concludes the paper and discusses future work.

II. Proposed Model

This works aims to propose a model for placement of DSP applications in a three-layer computing infrastructure that combines both public and private resources, as depicted in Fig. 1: IoT, which contains numerous geo-distributed resources (*e.g.*, sensors, IoT devices), often acting as data sources or sink actuators, but that can also be used for deploying DSP operators. The $Micro\ Datacenter$ layer also contains geo-distributed resources but with less stringent computational constraints than those in the IoT layer (*e.g.*, routers, gateways, and micro datacenters). The $Cloud$ contains high-end servers with fewer resource constraints [3].

IoT and $Micro\ Datacenter$ are grouped under the Edge computing paradigm. Hereafter the overall environment comprising the two-layer edge and cloud computing resources is termed as *cloud-edge infrastructure*. The following sections describe the Cloud-Edge data Stream Placement (CESP) model for addressing parallelism and placement of DSP applications onto cloud-edge infrastructure.

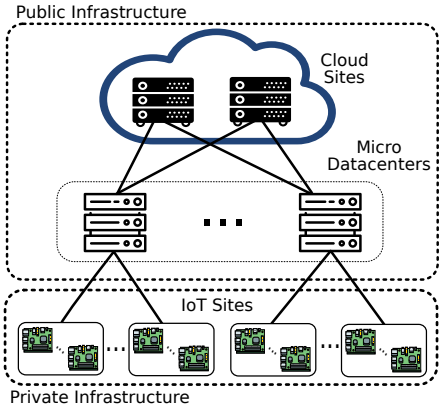


Fig. 1. Overview of the target computing infrastructure.

A. System Model

The cloud-edge infrastructure is represented as a graph $\mathcal{G}^I = \langle \mathcal{R}, \mathcal{P} \rangle$, where $\mathcal{R}$ is the set of computing resources of all layers ($\mathcal{R}^{IoT} \cup \mathcal{R}^{MD} \cup \mathcal{R}^{Cloud}$), and $\mathcal{P}$ is the set of network interconnections between computing resources. Table I summarizes the used notation.

A computing resource $k \in \mathcal{R}$ has CPU (CPU_k) and memory (Mem_k) capacities, given respectively in $100 \times num\_of\_cores$, and bytes. The processing speed of a resource (V_k) is its CPU clock in GHz. Similar to existing work [4], the network has a single interconnection between a pair of computing resources k and l, with bandwidth given by $Bw_{k,l}$ and latency by $Lat_{k,l}$.

TABLE I
Notation used in the paper.

Symbol	Description
$\mathcal{G}^I$ and $\mathcal{G}^A$	Graph of the infrastructure and requirements of a DSP application
$\mathcal{R}$	Set of computing resources ($\mathcal{R}^{IoT} \cup \mathcal{R}^{MD} \cup \mathcal{R}^{Cloud}$)
$\mathcal{P}$	Set of paths between computing resources
p	A path p that belongs to $\mathcal{P}$
p_s and p_d	Source and destination of path p
CPU_k and Mem_k	CPU and memory capacity of resource k
V_k	Clock speed of resource k
$Bw_{k,l}$ and $Lat_{k,l}$	Bandwidth and network latency of the path between resources k and l
$\mathcal{O}$ and $\mathcal{E}$	Set of operators and streams between operators
$Source^{\mathcal{O}}$, $Sink^{\mathcal{O}}$ and $Trans^{\mathcal{O}}$	Subset of sources, sinks and transformation operators from $\mathcal{O}$
AR^j and DR^j	Byte arrival and departure rate of operator j
S^j and C^j	Selectivity and data transformation factor of operator j
$\mathcal{U}^j$	Subset of operators that send data to j
$\rho^{i \to j}$	Probability that operator i sends data to operator j
Ref^j_{cpu} and Ref^j_{mem}	Ref. values of CPU and memory use for operator j to process Ref^j_{data}
Ref^j_{data}	Ref. data volume (bytes/s) processed to obtain Ref^j_{cpu} and Ref^j_{mem}
Req^j_{cpu} and Req^j_{mem}	CPU and memory requirements to process the data arriving at operator j
Ω_k	Speedup/slowdown w.r.t. the ref. clock V and the clock of resource k (V_k)
$x(j,l)$	Amount of data that operator j processes on l
$f(i,k \to j,l)$	Amount of data flowing from operator i to j deployed on resources k and l
$C_{cpu}(l)$ and $C_{mem}(l)$	Cost per CPU unit and cost of storing one byte in memory at resource l
$C_{bw}(k,l)$	Cost of transferring a byte over the network from resource k to l
CC and NC	Computational and network costs
ATT	Aggregate data transfer time
Ref^j_V	Processing speed of reference resource j
β	Safety margin in the processing requirements of an application

An application deployment request is a directed graph $\mathcal{G}^A = \langle \mathcal{O}, \mathcal{E} \rangle$, where $\mathcal{O}$ represents data source(s) $Source^{\mathcal{O}}$, data sink(s) $Sink^{\mathcal{O}}$ and transformation operators $Trans^{\mathcal{O}}$, and $\mathcal{E}$ represents the streams between operators, which are unbounded sequences of data (*e.g.*, messages, packets, tuples, file chunks) [1]. The application graph contains at least one data source, one transformation operator and one data sink.

An operator $j \in \mathcal{O}$ is a processing element given by the tuple $\langle S^j, C^j, \mathcal{U}^j, AR^j \rangle$, where S^j is the selectivity (message discarding percentage), C^j is the data transformation factor (how much it increases/decreases the size of arriving messages), $\mathcal{U}^j$ is the set of upstream operators directly connected to j, and AR^j is the input rate in Bps that arrives at the operator. When operator j is a data source (*i.e.*, $j \in Source^{\mathcal{O}}$) its input rate is the amount of data ingested into the application since $\mathcal{U}^j = \emptyset$. Otherwise, AR^j is recursively computed as:

$$AR^j = \sum_{i \in \mathcal{U}^j} \rho^{i \to j} \times DR^i \tag{1}$$

where $\rho^{i \to j}$ is the probability that operator i will send an output message to operator j, which captures how operator i

distributes its output stream among its downstream operators. DR^i is the departure date of operator i, given by:

$$DR^i = AR^i \times (1 - S^i) \times C^i \tag{2}$$

which is the size of the input stream after applying the selectivity S^i and the data transformation factor C^i.

A physical representation of the application deployment request graph is created when operators are placed onto available resources, as depicted in Figure 2. Operators placed within the same host communicate directly, whereas inter-resource communication is done via the *Data Transfer Service*. Messages that arrive at a computing resource are received by the *Dispatching Service*, which then forwards them to the destination operator within the computing resource. This service also passes messages to the Data Transfer Service when inter-resource communication is required. Each operator comprises an internal queue and a processing element, which are treated as a single software unit when determining the operator properties (*e.g.*, selectivity and data transformation factor), and its CPU and memory requirements. Moreover, an operator may demand more CPU than what a single resource can offer. In this case, multiple operator replicas are created in a way that each individual replica fits a computing resource.

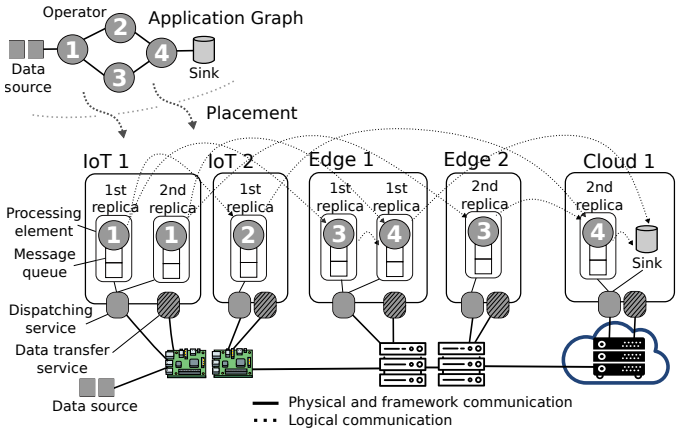


Fig. 2. Application graph adjusted to available resources (placement).

The quality of a placement depends on meeting the application requirements. In our work, the requirements of each operator j are expressed in terms of CPU (Req^j_{cpu}) and memory (Req^j_{mem}) that the operator needs to process its incoming byte stream.

One operator's CPU and memory requirements can be obtained by profiling it on a reference resource [5]. We call Ref^j_{cpu}, Ref^j_{mem} and Ref^j_{data} the reference values of operator j for CPU, memory and processed data, respectively. Since it is not possible to consume CPU and memory in factional numbers, we round up and combine reference values with AR^j of j, to compute CPU and memory requirements, *i.e.* Req^j_{cpu} and Req^j_{mem}, to handle its arriving data stream:

$$Req^j_{cpu} = \left\lceil \frac{Ref^j_{cpu} \times AR^j}{Ref^j_{data}} \right\rceil, Req^j_{mem} = \left\lceil \frac{Ref^j_{mem} \times AR^j}{Ref^j_{data}} \right\rceil \tag{3}$$

B. Problem Formulation

The problem is modeled as a MILP with variables $x(j,l)$ and $f(i,k \rightarrow j,l)$. Variable $x(j,l)$ accounts for the amount of bytes that a replica of operator j can process on resource l, whereas variable $f(i,k \rightarrow j,l)$ corresponds to the number of bytes that operator replica i on resource k sends to downstream operator replica j deployed on resource l.

CESP main goal is to minimize the deployment costs and the time to transfer the generated data over the network, both of which can impact the application performance. We consider that the rate of data ingested by the sources can be estimated and does not change over time. Moreover, resource usage incurs a cost. The cost of using one unit of CPU and storing one byte in memory at resource l is given by $C_{cpu}(l)$ and $C_{mem}(l)$, respectively. Based on Amazon Fargate's pricing scheme[1] we consider a large range of discrete values. The cost of transferring a byte over the network from resource k to l is denoted by $C_{bw}(k,l)$. The overall processing and network deployment costs of an application can be computed given the cost per resource unit.

As the processing requirements are computed based on a reference infrastructure and the combination of IoTs, MD and cloud resources produce a very heterogeneous set of resources, we apply a coefficient $\Omega_l = Ref^j_V / V_l$ to adapt the requirements for a resource l, in which Ref^j_V is the processing speed of the resource where reference values for operator j were obtained, and V_l is the clock speed of resource l. β refers to a safety margin to each replica requirements aiming for a steady state system. The computational cost incurred by the application is computed as:

$$CC = \sum_{l \in \mathcal{R}} \sum_{j \in \mathcal{O}} C_{cpu}(l) \times \frac{\frac{Req^j_{cpu}}{\Omega_l} \times \beta \times x(j,l)}{AR^j} + C_{mem}(l) \times \frac{Req^j_{mem} \times x(j,l)}{AR^j} \tag{4}$$

The network cost considers the network unit cost and amount of data transferred through any path p that crosses link a,b. Resources at extremities of path p hosting replicas i and j, respectively, are p_s and p_d. Then the network costs is:

$$NC = \sum_{p \in \mathcal{P}} \sum_{a,b \in p} \sum_{j \in \mathcal{O}} \sum_{i \in \mathcal{U}^j} C_{bw}(a,b) \times f(i, p_s \rightarrow j, p_d) \tag{5}$$

Aiming to compose a multi-objective optimization function with variables from heterogeneous domain values, CESP applies a normalization to obtain values between 0 and 1. An overall cost is computed by combining both computation and network costs, and then it is normalized as follows:

$$C = \frac{(CC + NC)}{\max C_{cpu} + \max C_{mem} + \max C_{bw}} \tag{6}$$

[1]https://aws.amazon.com/fargate/pricing

The Aggregate Data Transfer Time (ATT) sums up the network latency of a link and the time to transfer all the data crossing it, and is normalized by the time it takes to send an amount of data that fills up the link capacity:

$$ATT = \sum_{p \in \mathcal{P}} \sum_{k,l \in p} \sum_{j \in \mathcal{O}} \sum_{i \in \mathcal{U}^j} \frac{f(i, p_s \rightarrow j, p_d) \times (Lat_{k,l} + \frac{1}{Bw_{k,l}})}{Lat_{k,l} + 1} \tag{7}$$

Treating the metrics equally – *i.e.*, ATT and C– the objective function aims to minimize the data transfer time and the application deployment costs:

$$\min : ATT + C \tag{8}$$

The objective function is subject to:

Physical constraints: The requirements of each operator replica j on resource l are a function of $x(j,l)$; *i.e.*, a fraction of the byte rate operator j should process (AR^j) plus the safety margin (β). The processing requirements of all operator replicas deployed on resource l must not exceed its processing capacity.

$$CPU_l \geq \sum_{j \in \mathcal{O}} \frac{\frac{Req^j_{cpu}}{\Omega_l} \times \beta \times x(j,l)}{AR^j} \qquad \forall l \in \mathcal{R} \tag{9}$$

$$Mem_l \geq \sum_{j \in \mathcal{O}} \frac{Req^j_{mem} \times x(j,l)}{AR^j} \qquad \forall l \in \mathcal{R} \tag{10}$$

There are also physical limitations imposed by the network infrastructure. If the amount of data crossing a link exceeds its bandwidth capacity, some delay is introduced due to network packet queuing. The amount of data crossing every link a, b must not exceed its bandwidth capacity.

$$\sum_{j \in \mathcal{O}} \sum_{i \in \mathcal{U}^j} f(i, p_s \rightarrow j, p_d) \leq Bw_{a,b} \qquad \forall a, b \in p; \forall p \in \mathcal{P} \tag{11}$$

Processing constraint: The amount of data processed by all replicas of j must be equal to the byte arrival rate of j.

$$AR^j = \sum_{l \in \mathcal{R}} x(j,l) \qquad \forall j \in \mathcal{O} \tag{12}$$

Flow constraints: Except for *sources* and *sinks*, it is possible to create one replica of operator j per resource, although the actual number of replicas, the processing requirements, and the interconnecting streams are decided within the model. Equation 13 ensures that the amount of data that flows from all replicas of i to all the replicas of j is equal to the departure rate of upstream i to j.

$$DR^i \times \rho^{i \rightarrow j} = \sum_{k \in \mathcal{R}} \sum_{l \in \mathcal{R}} f(i, k \rightarrow j, l) \qquad \forall j \in \mathcal{O}; \forall i \in \mathcal{U}^j \tag{13}$$

In this way, the amount of data flowing from one replica of i can be distributed among all replicas of j:

$$x(i,k) \times (1 - S^i) \times C^i \times \rho^{i \rightarrow j} = \sum_{l \in \mathcal{R}} f(i, k \rightarrow j, l) \qquad \forall k \in \mathcal{R}; \forall j \in \mathcal{O}; \forall i \in \mathcal{U}^j \tag{14}$$

On the other end of the flow, the amount of data that flows from all the replicas of all upstream operators i to each replica of j must be equal to the amount of data processed in $x(j,l)$:

$$\sum_{i \in \mathcal{U}^j} \sum_{k \in \mathcal{R}} f(i, k \rightarrow j, l) = x(j,l) \qquad \forall j \in \mathcal{O}; \forall l \in \mathcal{R} \tag{15}$$

Domain constraints: The placement k of sources and sinks is fixed and provided in the deployment requirements. Variables $x(j,l)$ and $f(i, k \rightarrow j, l)$ represent respectively the amount of data processed by j in l, and the amount of data sent by replica i in k to replica j in l. Therefore the domain of these variables is a real value greater than zero:

$$x(j,l) = AR^j \qquad \forall j \in Source^{\mathcal{O}} \cup Sink^{\mathcal{O}}; \forall l \in \mathcal{R} \tag{16}$$

$$x(j,l) \geq 0 \qquad \forall j \in Trans^{\mathcal{O}}; \forall l \in \mathcal{R} \tag{17}$$

$$f(i, k \rightarrow j, l) \geq 0 \qquad \forall k, l \in \mathcal{R}; j \in \mathcal{O}; i \in \mathcal{U}^j \tag{18}$$

III. Performance Evaluation

This section describes the experimental setup, the price model for computing resources, and performance evaluation results.

A. Experimental Setup

The solution is evaluated via discrete-event simulation using a framework built on OMNET++ to model and simulate DSP applications. The model is solved using CPLEX v12.9.0. The infrastructure comprises 105 resources: 35 IoT resources, 35 MD servers, and 35 cloud servers. The resource capacity is modeled according to the characteristics of DSP applications and the layer in which the resource is located. IoT resources are modeled as Raspberry Pi's 3 (*i.e.,* 1 GB of RAM, 4 CPU cores at 1,2 GHz). As DSP applications are often CPU and memory intensive, the selected MD and cloud resources should be optimized for such cases. The offerings for MDs are still fairly recent. Existing work highlights that the choices of MD resources are more limited than those of the cloud, with more general-purpose resources. In an attempt to use resources similar to those available on Amazon EC2, MD resources are modeled as general-purpose t2.2xlarge machines (*i.e.,* 32 GB of RAM, 8 CPU cores at 3.0 GHz). Cloud servers are high-performance C5.metal machines (*i.e.,* 192 GB of RAM, 96 CPU cores at 3.6 GHz).

Resources within a site communicate via a LAN, whereas IoT sites, MDs, and cloud are interconnected by a single WAN path. The LAN has a bandwidth of 100 Mbps and 0.8 ms of latency. The WAN bandwidth is 10 Gbps and is shared on the path from the IoT to the MD or to the cloud, and the latency from IoT is 20 ms and 90 ms to the MD and cloud,

respectively. The latency values are based on those obtained by empirical experiments carried out by *Hu et. al* [4].

To evaluate CESP considering diverse and generic applications, we crafted multiple application graphs with various shapes and sizes. Existing work evaluated application graphs of several orders and interconnection probabilities, usually assessing up to 3 different graphs [1], [6]–[8]. Using a built-in-house python library, we built five graphs to mimic the behavior of large DSP applications. The graphs have various shapes and data replication factors for each operator, as depicted in Fig. 3. The applications have 25 operators, often more than what is considered in the literature [9]. They also have multiple sources, sinks, and paths, similar to previous work by *Liu and Buyya* [8]. As the present work focuses on IoT scenarios, the sources are placed on IoT resources, and sinks are uniformly and randomly distributed across layers as they can be acting as actuators – except for one sink responsible for data storage, which is placed in the cloud.

The operator properties are based on the RIoTBench [10]; an IoT application benchmark that offers 27 operators common to IoT applications and 4 datasets with IoT data. The experiments use the CITY dataset with 380 byte messages collected every 12 seconds containing environmental information (temperature, humidity, air quality) from 7 cities across 3 continents. It has a peak rate of 5000 tuples/s, which is continuous and divided among sources. The remaining properties are drawn from the values in Table II.

The Req^j_{cpu} of an operator j can be computed based on measurements obtained via application profiling, including Ref^j_{cpu} and Ref^j_{data}, using techniques proposed in existing work [5]. In practice, the arrival byte rate AR^j, the probability that an upstream operator i sends data to j, *i.e.* $\rho^{i \to j}$, selectivity S^j, and data transformation pattern C^j could be average values obtained via application profiling. However, to create a worst-case scenario in terms of load, $\rho^{i \to j}$ is set to 1 for all streams in each application request. As CESP creates multiple replicas, $\rho^{i \to j}$ gets divided among instances of operator j, hence creating variations on the arrival rate of downstream operators during runtime. The operator processing requirements estimated by the model may not be enough to handle the actual load during certain periods, so resulting in large operator queues. To circumvent this issue, we add a small safety margin, the β factor, which is a percentage increase in the application requirements estimated by CESP. A β too high results in expensive over-provisioning. We evaluated multiple values of β and set it to 10%, which gives a performance boost to handle the queues without incurring high costs.

Price model: The price for resources is derived from Amazon AWS services, considering the US East Virginia location. The CPU and memory prices are computed based on the AWS Fargate Pricing[2] under a 24/7 execution. We consider a Direct Connection[3] for the network between the IoT site and the AWS infrastructure. As DSP applications generate large amounts of data, we consider a Direct Connection of 10 GB/s. The data sent from IoT sites to AWS infrastructure uses AWS IoT Core[4]. Connections between operators either on MD or IoT resources to the cloud use Private Links[5]. Amazon provides the values for CPU, memory, and network as, respectively, a fraction of a vCPU, GB, and Gbps, but in our formulation, the values for the same metrics are computed in CPU units ($100 * num\_cores$), bytes and Mbps. The values provided by Amazon, but converted to the scale used in our formulation are presented in Table III. As the environment combines both public and private infrastructure, deployment costs are applied only to MD and cloud resources, the network between these two, and the network between these two and IoT resources. The communication between IoT resources is free since they are on the same private network infrastructure.

TABLE II
OPERATOR PROPERTIES IN THE APPLICATION GRAPHS.

Property	Value	Unit
Selectivity	0 - 20	%
Data Transformation Factor	70 - 130	%
Reference CPU	1 - 26	CPU units
Reference Memory	1 - 27300000	bytes
Reference Data	38 - 2394000	bytes

TABLE III
COMPUTING AND NETWORK COSTS.

Resource	Unit	Cost
CPU	CPU/month	$0.291456
Memory	byte/month	$3.2004e-09
Direct Link IoT to AWS	10GB link/Month	$1620
Link IoT to AWS	Connection/Month	$0.003456
	KB	$0.0000002
Communication IoT to cloud, IoT to MD, and MD to cloud	GB	$7.2 + 0.01 per GB

Evaluated approaches and metrics: Five different configurations of deployment requests are submitted for each application during 120 simulated seconds. The reported values for each application are averages of these five executions. Each request has a different placement for sources and sinks, but still respecting the rule of sources always on IoT resources and at least one sink in the cloud. The operator properties such as selectivity and data transformation factor vary across configurations. Aiming to investigate the benefits of using both IoT and MD resources, we consider two CESP versions: one that considers all resources from the infrastructure (**CESP–All**) and another that considers only IoT and cloud resources (**CESP–IC**). Both implementations of the model are compared against a traditional approach called **Cloud-Only** that applies a random walk considering only cloud resources.

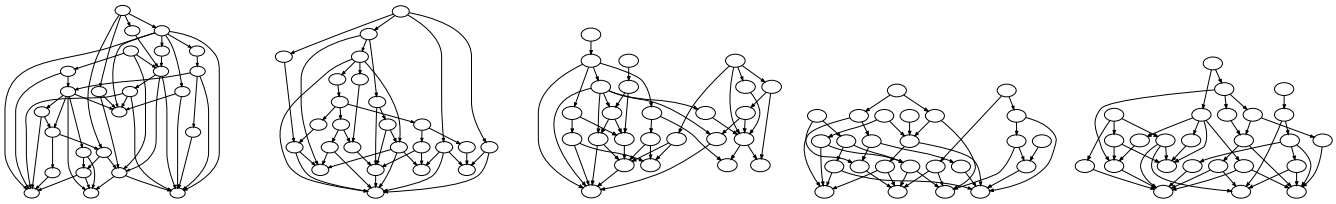
As for performance metrics, we consider the *throughput*, which is the processing rate, in bytes/s, of all sinks in the

[2]https://aws.amazon.com/fargate/

[3]https://aws.amazon.com/directconnect/

[4]https://aws.amazon.com/iot-core/

[5]https://aws.amazon.com/privatelink/



(a) App 1 (b) App 2 (c) App 3 (d) App 4 (e) App 5

Fig. 3. Application graphs used in the evaluation.

application; and *end-to-end latency*, which is the average time span between the generation until the message reaches a sink. CESP takes the throughput into account in the constraints and the end-to-end latency indirectly by optimizing the ATT.

B. Performance Results

Throughput results are summarized in Fig. 4(a). Under most scenarios, Cloud-Only and CESP achieve similar throughput. However, under App1 and App2, Cloud-Only performs much worse than both CESP versions because of the depth of the application graphs and the fact that they have many splitter operators in early stages, close to data sources. Given that $\rho^{i \to j} = 1$, the number of messages quickly increases as events reach the splitters, and that results in a large amount of data traversing the rest of the graph. That becomes an issue to Cloud-Only, while both CESP versions are able to cope with this scenario better. Apps 3-5, on the other hand, have sources spread across many resources and shallower graphs, which Cloud-Only can handle similar to CESP.

Processing data only in the cloud has a negative effect on end-to-end latency, as shown in Figure 4(b). The network becomes a bottleneck, especially in the LAN sections. Messages are queued, producing a high end-to-end latency for Cloud-Only, even in scenarios where Cloud-Only had similar throughput. CESP tackles this network problem by placing communicating operators closer to one another in terms of network latency – *i.e.*, placing sources and their immediate downstream operators on the same resource. As IoT resources are computationally constrained, placing communicating operators on the same device becomes challenging. CESP breaks an operator into small replicas, thus allowing this co-placement. Even if the replica processes only part of an operator's load, it helps by reducing the data sent through the network, hence reducing congestion. With this process of co-placement and operator replication, CESP reduces the end-to-end latency by at least 80%

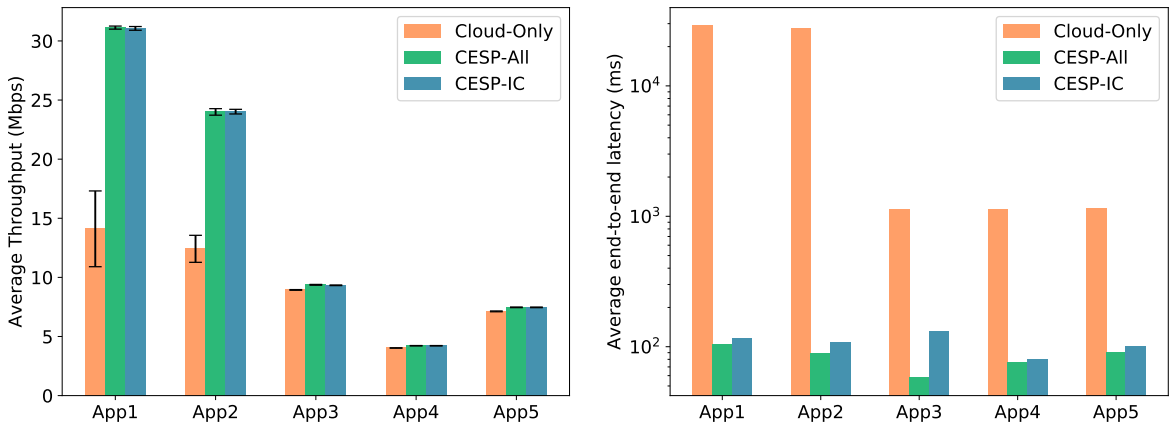
Table IV contains the average CPU and memory requirements per operator instance for the evaluated DSP applications. As Cloud-Only does not create replicas, its reported values are the average requirements per operator, computed as the one used by both CESP implementations. Results demonstrate that CESP divides the operator into multiple replicas, each of which has significantly smaller requirements allowing for better utilization of IoT and MD resources with co-placement at the edges of the network hence experiencing lower network latency. By breaking an operator into replicas with lower requirements and using IoT and MD resources, CESP achieves better end-to-end latency and deployment costs.

TABLE IV
AVERAGE RESOURCE CONSUMPTION PER OPERATOR INSTANCE.

	Cloud-Only		**CESP-All**		**CESP-IC**	
	CPU (%)	**Memory (bytes)**	**CPU (%)**	**Memory (bytes)**	**CPU (%)**	**Memory (bytes)**
App1	436.1760	8177966180	0.0018	3202	0.0019	3462
App2	550.5440	10979496789	0.0006	2790	0.0007	2521
App3	411.9440	4325880180	0.0012	4099	0.0023	3014
App4	390.2320	1861834684	0.0043	7895	0.0044	8691
App5	430.4480	6167321808	0.0028	7161	0.0030	6950

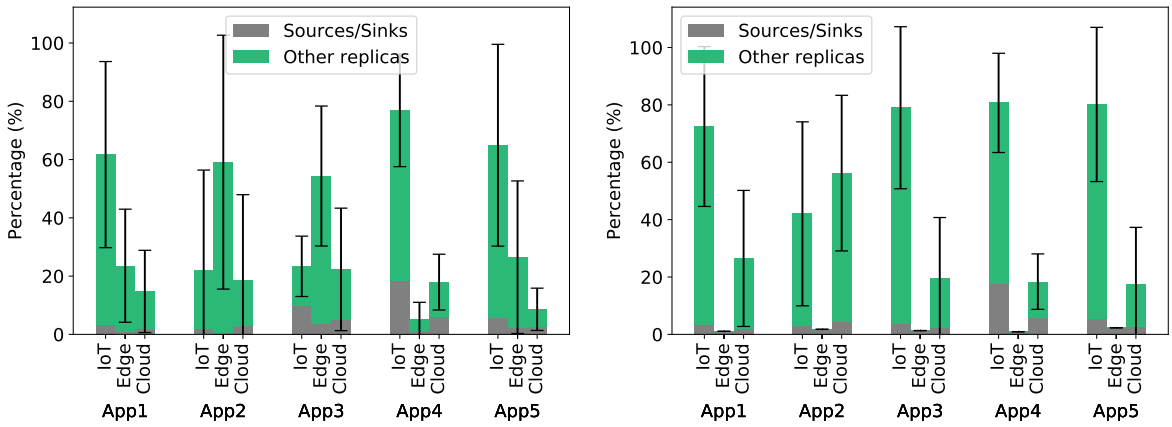
Fig. 5 shows the percentage of replicas deployed in each layer. The bottom part of each bar presents the percentage of sources and sinks deployed in the respective layer, whereas the top part corresponds to other operators. CESP–All provides better end-to-end latency than CESP–IC due to the use of MD resources. MD resources are computationally more powerful than IoT resources, so enabling the co-placement of more replicas on the same node. When the volume of data processed by each operator grows, it becomes inefficient to create several replicas on IoT resources. CESP creates multiple replicas of communicating operators as pipelines and places each pipeline in a different device, resulting in the end-to-end latency improvement of CESP–All when compared with CESP–IC. As CESP–IC does not use MD resources, it continues to explore IoT resources, by creating multiple replicas combined into pipelines into different devices. The downside of using IoT resources for this is that the pipeline is shorter, requiring the use of the network to communicate with the downstream replica, resulting in higher end-to-end latency.

Fig. 6 shows the deployment costs. Cloud-Only does not consider IoT resources, which are free of charge, and does not reduce the amount of data traversing the Internet. It yields the highest deployment cost, both computational and network. Along with the end-to-end latency gain from co-placing small replicas into IoT resources, CESP–IC explores such devices as they are free of charge. CESP–All, which explores MDs in addition to IoT resources, is able to deploy more replicas at the edges of the network, thus reducing the network usage and costs. CESP–All experiences the cheaper deployment costs.



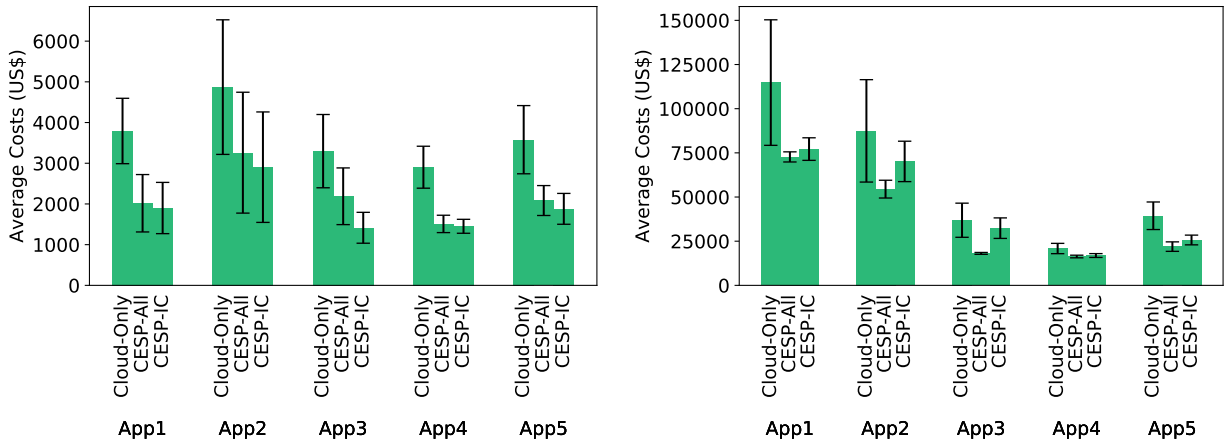
(a) Throughput (b) End-to-end latency

Fig. 4. Throughput and end-to-end latency under Cloud-Only and CESP.



(a) CESP–All (b) CESP–IC

Fig. 5. Replica distribution per resource for both CESP versions.



(a) Computational costs (b) Network costs

Fig. 6. Computational and network costs under Cloud-Only, CESP–All and CESP–IC.

IV. Related Work

Traditionally DSP frameworks such as Apache Storm, Apache Flink, among others were conceived to run on clusters of homogeneous resources and on cloud infrastructure. To

minimize the end-to-end latency of processing events and the amount of traffic that traverses public networks, existing work has introduced novel DSP architecture for placing certain operators or tasks at the edges of the Internet, often closer to where the data is generated [11].

The problem of placing DSP dataflows onto heterogeneous resources has been shown to be at least NP-Hard [2]. Moreover, most of the existing work neglects the communication overhead [12], although it is relevant in geo-distributed infrastructure [4]. Likewise, the considered applications are often oversimplified, ignoring operator patterns such as selectivity and data transformation [13].

Effort has been made on modelling the operator placement on cloud-edge infrastructure, including sub-optimal solutions [14], [15] while others focus on end-to-end latency while neglecting throughput, application deployment costs, and other performance metrics when estimating the operator placement [1], [16]. There are also solutions on models that ignore operator parallelism [17], which is an issue when dealing with *IoT* constrained resources. Existing work also explores Network Function Virtualization (NFV) for placing *IoT* application service chains across cloud-edge infrastructure [18]. Solutions for profiling and building performance models of DSP operators are also available [5].

Although the dataflow model followed by several DSP solutions shares many similarities with workflow scheduling [19], it presents unique challenges related to handling unbounded streams of data that can have varying load, and the characteristics of operators found in DSP graphs.

This work addresses operator placement and parallelism across cloud-edge resources considering computing and communication constraints. The solution offers a framework to model both issues as an optimal MILP problem and solve it by optimizing the aggregate data transfer time, and application deployment costs, considering the pricing scheme of a major infrastructure provider.

V. Conclusion

This paper presented CESP, a MILP formulation for the operator placement and parallelism of DSP applications that optimizes the end-to-end latency and deployment costs. CESP combines profiling information with the computed amount of data that each operator should process, to obtain the processing requirements so that the operator can handle the arriving load. CESP also creates multiple lightweight replicas to offload operators from the cloud to the edges of the network hence obtaining lower end-to-end latency.

Two versions of CESP were evaluated using various applications with different configurations in terms of selectivity, data transformation pattern, and CPU and memory requirements. Both versions provide at least $\simeq 1\%$ throughput improvement, $\simeq 80\%$ end-to-end latency reduction and deployment costs $\simeq 30\%$ cheaper than a traditional placement scheme. The results also show that by using MD resources, the end-to-end latency can be improved by at least $\simeq 6\%$ and deployment costs reduced by $\simeq 4\%$. Future work aims to improve the solution robustness by relaxing assumptions on constant event arrival rate and static IoT infrastructure.

References

[1] 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.

[2] 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.

[3] D. Puthal, M. S. Obaidat, P. Nanda, M. Prasad, S. P. Mohanty, and A. Y. Zomaya, "Secure and sustainable load balancing of edge data centers in fog computing," *IEEE Communications Magazine*, vol. 56, no. 5, pp. 60–65, 2018.

[4] 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 *Proc. of the 7th ACM SIGOPS Asia-Pacific Workshop on Systems*. ACM, 2016, p. 5.

[5] 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 2019)*, Brno, Czech Republic, Mar. 2020.

[6] T. Hiessl, V. Karagiannis, C. Hochreiner, S. Schulte, and M. Nardelli, "Optimal placement of stream processing operators in the fog," in *2019 IEEE 3rd Int. Conf. on Fog and Edge Computing (ICFEC)*. IEEE, 2019, pp. 1–10.

[7] B. Gedik, S. Schneider, M. Hirzel, and K.-L. Wu, "Elastic scaling for data stream processing," *IEEE Tr. on Parallel and Distributed Systems*, vol. 25, no. 6, pp. 1447–1463, 2013.

[8] X. Liu and R. Buyya, "Performance-oriented deployment of streaming applications on cloud," *IEEE Tr. on Big Data*, vol. 5, no. 1, pp. 46–59, March 2019.

[9] S. Zeuch, B. D. Monte, J. Karimov, C. Lutz, M. Renz, J. Traub, S. Breß, T. Rabl, and V. Markl, "Analyzing efficient stream processing on modern hardware," *Proc. VLDB Endow.*, vol. 12, no. 5, pp. 516–530, Jan. 2019.

[10] A. Shukla, S. Chaturvedi, and Y. Simmhan, "Riotbench: A real-time iot benchmark for distributed stream processing platforms. corr abs/1701.08530 (2017)," *arxiv. org/abs/1701.08530*, 2017.

[11] V. Cardellini, V. Grassi, F. Lo Presti, and M. Nardelli, "Optimal operator placement for distributed stream processing applications," in *Proc. of the 10th ACM Int. Conf. on Distributed and Event-based Systems*. ACM, 2016, pp. 69–80.

[12] B. Cheng, A. Papageorgiou, and M. Bauer, "Geelytics: Enabling on-demand edge analytics over scoped data sources," in *IEEE Int. Cong. on BigData*, 2016.

[13] 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.

[14] 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 Mgmt (IM)*, May 2017.

[15] W. Chen, I. Paik, and Z. Li, "Cost-aware streaming workflow allocation on geo-distributed data centers," *IEEE Transactions on Computers*, Feb 2017.

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

[17] E. Gibert Renart, A. Da Silva Veith, D. Balouek-Thomert, M. D. De Assunção, L. Lefèvre, and M. Parashar, "Distributed operator placement for IoT Data Analytics Across Edge and Cloud Resources," in *2019 19th IEEE/ACM Int. Symp. on Cluster, Cloud and Grid Computing (CCGRID)*, May 2019, pp. 459–468.

[18] D. T. Nguyen, C. Pham, K. K. Nguyen, and M. Cheriet, "Placement and chaining for run-time IoT service deployment in edge-cloud," *IEEE Transactions on Network and Service Management*, pp. 1–1, 2019.

[19] G. L. Stavrinides, F. R. Duro, H. D. Karatza, J. G. Blas, and J. Carretero, "Different aspects of workflow scheduling in large-scale distributed systems," *Simulation Modelling Practice and Theory*, vol. 70, pp. 120–134, 2017.