Scheduling Continuous Operators for IoT edge Analytics with Time Constraints

Patient Ntumba, Nikolaos Georgantas, Vassilis Christophides

Scheduling of Continuous Operators for IoT edge Analytics with Time Constraints

Patient Ntumba
Inria Paris, France
patient.ntumba@inria.fr

Nikolaos Georgantas
Inria Paris, France
nikolaos.georgantas@inria.fr

Vassilis Christophides
ENSEA, ETIS, France
vassilis.christophides@ensea.fr

***Abstract*—Data stream processing and analytics (DSPA) engines are used to extract in (near) real-time valuable information from multiple IoT data streams. Deploying DSPA applications at the IoT network edge through Edge/Fog architectures is currently one of the core challenges for reducing both network delays and network bandwidth usage to reach the Cloud. In this paper, we address the problem of scheduling continuous DSPA operators to Fog-Cloud nodes featuring both computational and network resources. We are paying particular attention to the dynamic workload of these nodes due to variability of IoT data stream rates and the sharing of nodes' resources by multiple DSPA applications. In this respect, we propose TSOO, a resource-aware and time-efficient heuristic algorithm that takes into account the limited Fog computational resources, the real-time response constraints of DSPA applications, as well as, congestion and delay issues on Fog-to-Cloud network resources. Via extensive simulation experiments, we show that TSOO approximates an optimal operators' placement with a low execution cost.**

***Index Terms*—IoT (Edge), data stream, Fog, Cloud, continuous operator, scheduling, real time, queuing model**

I. Introduction

The Internet of things (IoT) delivers at real-time massive amounts of data. To support (near) real-time control and automation applications such as smart transportation [1] or security [2], distributed IoT data streams need to be processed and analysed on the fly. Thus, *data stream processing and analytics* (DSPA) engines are used to extract in real-time valuable information from unbounded data streams via a series of continuous operators (i.e. aggregation, filter, join, etc.) [3].

Figure 1 highlights a DSPA application for country-wide traffic monitoring, as well as, city-wide traffic regulation. The latter enables control of traffic lights [1] e.g., to give priority to jammed traffic flows over non-jammed ones. We need to specify DSPA operators that analyse the data streams produced by the moving vehicles (e.g., vehicle identifier, GPS-location, driving speed). These operators are usually deployed at the Cloud where IoT data streams are transmitted e.g., by 5G based street antennas via several hops of the wide area network (WAN) [4]. At high data stream rates, the WAN bandwidth to reach the Cloud can become a bottleneck as it may incur high network delays that may impair on the real-time constraint of DSPA applications. Furthermore, the WAN bandwidth is inherently dynamic due to the varying internet traffic conditions [5]. When the sum rate of transmissions of traffic data from vehicles are overwhelming the bandwidth capacity agreed between the Cloud provider and ISPs, the monetary cost charged for the WAN bandwidth usage can be much larger than the cost for computational resource usage [6].

Deploying DSPA operators closer to where IoT data streams are produced through Edge/Fog computing is currently one of the core challenges for reducing both network delays and bandwidth compared to DSPA applications deployed exclusively on the Cloud [7]. The Edge/Fog computing introduces intermediate layers of computing, (storage), and networking between the IoT devices and the Cloud. The two terms are often used interchangeably. However, according to [8], we consider them as different layers: Edge computing is the network layer encompassing the IoT devices and their users, to provide, for example, local computing capability on a sensor (e.g. smart phone, Raspberry Pi, connected vehicle, drone, etc.). Fog computing residing between IoT devices network (Edge) and central Cloud, consists of Fog nodes which are either physical or virtual components (e.g. routers, small server, cloudlet, etc.). However, Edge/Fog nodes come with limited resources both in terms of computational power (e.g., CPU/GPU, RAM, etc.) and power supply (e.g., batteries, etc.). Thus, we need to schedule DSPA operators across Edge-Fog-Cloud resources by controlling both the processing latency and the network delays while keeping minimal the cost of using these resources [9].

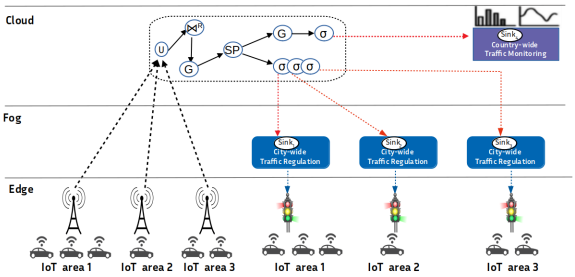


Fig. 1: Traffic management application adapted from [1]

For instance, in our motivating application, traffic data rates may significantly vary per geographic area due to the high variability of the traffic density, as well as, of the mobility patterns. Clearly, workload picks require a higher usage of computational resources, energy and bandwidth and may incur higher network delays and processing latency [9]. Therefore, a dynamic scheduling at run time is needed in order to decide how/when to schedule DSPA operators w.r.t. the evolution of the workload [10]. In our previous work [11], we introduced a cost model of both network (bandwidth, delay) and computational resources (CPU/RAM) of network nodes. Using this model, we formulated the problem of scheduling DSPA operators between the Fog and the Cloud nodes to process data streams produced by mobile IoT devices at the Edge as a single objective optimization (SOO) problem of the combined resource cost subject to the constraints of the computational resource usage, cloud bandwidth usage and operator replicability constraint. As SOO is a NP-hard problem, we introduced a greedy algorithm that approximates an initial optimal placement of operators to replicate from the Cloud to the Fog. However, SOO did not address dynamic re-scheduling of continuous operators that also satisfies real-time response constraints.

In this paper, we extend SOO with real-time response constraints, that we formulate as a Time based Single Objective Optimization (TSOO) problem. We show that this problem is NP-hard and hence, the contributions of this paper are the following:

- we define the response time of a DSPA application by abstracting each operator by a queuing model [12];

- we formalize TSOO to dynamically allocate both computational (i.e., CPU/RAM) and network resources (i.e., network delays and bandwidth);
- we introduce a framework for monitoring the resource usage of network nodes as well as the workload of a DSPA application in order to trigger dynamic operator scheduling in the Edge-Fog-Cloud continuum;
- we propose a heuristic algorithm called TSOO to approximate the best feasible scheduling of operators at run-time.

We experimentally evaluate TSOO by using the simulation tool iFogSim [13] while considering heterogeneous computational and network resources across Edge-Fog-Cloud architecture and IoT data streams with varying rates. As baseline we consider TRCS [14] that improves a pure cloud based processing by using as less as possible Fog resources to ensure Cloud bandwidth usage constraint and the real-time response constraint. We also compare TSOO with state of the art methods based on simulated annealing (SA) (TSOO-SA) [15]. We use SA because of its flexibility and global optimization capabilities to approximate the global optimum of NP-hard problems. Furthermore, we use the heuristic algorithm proposed by Rizou et al. [16] that minimizes only the network resource usage cost while satisfying a real-time response constraint.

Our simulation results show that TSOO approximates well (with only 5% difference ratio) the optimal solution produced by TSOO-SA of using computational and network resources both statically and at run-time. However, TSOO-SA produces an operator placement which is not likely to satisfy real-time response constraints. Unlike TSOO-SA solutions that may end-up with 100% of constraint violations, TSOO solutions violates only up to 6.6% the real-time response constraint in the worst case of high rate data streams.

The remainder of this paper is organized as follows: Section II introduces our core model of DSPA applications and formulates the resource allocation problem. Section III extends our core model with real-time response constraints that DSPA applications have to satisfy. Section IV presents our heuristic algorithm TSOO for approximating optimal placements of DSPA operators. Section V details the experimental evaluation. Finally, Sections VI and VII present respectively the related work and the conclusions.

II. Background

In this section we present the core model of Edge-Fog-Cloud architectures and DSPA applications and we formulate the resource allocation problem (for more details, readers are referred to [11]).

A. Edge-Fog-Cloud Architecture

We abstract an Edge-Fog-Cloud architecture as a hierarchical WAN of resources defined by the set H=$\{E, F, C\}$ [17]. The Edge (E) layer consists of M IoT devices E=$\{E_1, ..., E_M\}$ moving in N geographic areas A_j, j=$\{1...N\}$, the Fog (F) layer consist of N Fog nodes F=$\{F_1, ..., F_N\}$ where each Fog node F_j provides nearby computational service to the geographic area A_j. One Cloud node C is considered at the top of the hierarchy.

Edge-Fog-Cloud nodes are characterized by the following parameters: (i) cm_{E_i} the available CPU/memory of an Edge node E_i and cm_{F_j} of a Fog node F_j; (ii) $nd_{E_iF_j}$ the network delay of an Edge node E_i to its nearest Fog node F_j; (iii) nb_{EF_j} the available network bandwidth capacity of all incoming links to the fog node F_j; (iv) nd_{F_jC} the network delay of the link from a Fog node F_j to the Cloud node C; (v) nb_{F_jC} the available network bandwidth capacity on the Fog node F_j to Cloud network link. The CPU/memory of the Cloud node C, cm_C, is practically considered unlimited [17].

We additionally consider the usage of CPU/memory cmu_{E_i}, cmu_{F_j} and cmu_C respectively on a Edge node E_i, a Fog node F_j and the Cloud node C, as well as the usage of the network bandwidth $nbu_{E_iF_j}$ and nbu_{F_jC} on links respectively from an Edge node E_i to a Fog node F_j and from a Fog node F_j to the Cloud node C. Finally, let S_j be the sum of data streams arriving to a Fog nodes F_j and produced by $m_j(t)$ IoT devices moving at a time t in a geographic area A_j. Hence, $M = \sum_{j=1}^{N} m_j(t)$.

B. DSPA application model

We model a DSPA application as a directed acyclic graph (DAG) of operators, denoted by G , where the vertices represent the operators and the edges the data stream flowing between two operators [3]. G topology further includes the sources that produce the raw data streams S_j of rate $|S_j|$ consumed by DSPA operators and sinks that capture the stream of the computed results. To cope with the infinite nature of data streams, we consider that continuous operators are executed in time windows ω_x to process a finite set of data items d_x arising within a time interval.

The operators are characterized by the following parameters: i) sel_x: the operator *selectivity* defined as the ratio between the input and output data rate of an operator O_x; ii) $\lambda_{x,y}$: the *edge data rate* defined as the rate of data stream flow between two operators; iii) c_x: the computational cost of an operator [18] defined in terms of CPU (e.g., million instructions (MI) demand per byte of data) and/or RAM (e.g., 1Mb demand per 10Mb of data) for an operator O_x to process a unitary data d_x; and iv) D_x: the operator data load defined as the aggregation of the input data items d_x per time window ω_x: D_x=$\sum_{i=1}^{I} \omega_x \cdot \lambda_{i,x}$. Where I is the maximum number of upstream operators O_x producing data stream at rate $\lambda_{i,x}$ towards the operator O_x. Finally, we denote by $req_x = D_x \cdot c_x$ the computational resources CPU/memory required by an operator O_x given its data load D_x at a time t and its cost c_x.

To replicate and migrate a part of G in different computational nodes $n_i = E_i|F_j|C$ of an Edge-Fog-Cloud architecture, we need to partition G in disjoint subgraphs,$Gmig_i$, according to some workload criteria. The resulting graph to deploy is defined as follows:

$$G_{dep} = \bigcup_{\forall_{n_i \in H}} Gmig_i \tag{1}$$

To specify a replication and migration point in G, we rely on the *edge-cut* algorithm which partitions G in two disjoint subgraphs [14]. An edge-cut ec_j contains the set of edges having one endpoint in each subgraph of the partition. Let $|ec_j|$ be the rate of an edge-cut ec_j defined as the sum of edge data rates crossing this edge-cut.

C. Resource allocation problem

To capture the fact that computational resources allocated to a DSPA application at different layers of an Edge-Fog-Cloud architecture are limited, we weight the computational resource usage of different operators sharing a node by the inverse of the available computational resources at this node. Our model relies on the following intuitive assumptions [11], [17]: (i) the computational resources are practically unlimited in the Cloud, $cm_C \rightarrow \infty$, thanks to the on-demand resource scaling, and hence the weight in the cloud is practically zero, $\frac{1}{cm_C} \rightarrow 0$; (ii) the computational resources of Fog and Edge nodes are limited as they can not be scaled on demand, and thus the weight in a Fog (or Edge) node is non-zero, $\frac{1}{cm_{F_j}} \in]0, 1]$ (or, $\frac{1}{cm_{E_i}} \in]0, 1]$). To simplify our model, we consider that mobile IoT devices at the Edge layer are not used for processing data streams

[17]. This will be address in the future work. Then, the focus of our work is to optimize the Fog computational resource usage cost:

$$cru = \sum_{j=1}^{N} cmu_{F_j} \cdot \frac{1}{cm_{F_j}} \quad (2)$$

cmu_{F_j} is the sum of the RAM (or CPU/GPU) usage required (req_x) by each operator of a subgraph $Gmig_j$, to replicate on F_j.

By abstracting the Edge-Fog-Cloud architecture as a hierarchical WAN of resources, two conflicting factors are involved [7]: (i) the network bandwidth is increasing in the hierarchy as we go from the Edge to the Cloud and (ii) the network delay that also increases up as the Cloud is in two hops from the Edge and the Fog at one hop.

Unlike [11], we consider the fact that the network delays and network bandwidths of each individual WAN links can be dynamic with regard to the network conditions [5]. Thus, we need to differentiate network links not only according to their delay [19] but also by their available bandwidth [11]. In this respect, the bandwidth usage of a network link is weighted by the inverse of its available bandwidth multiplied by the ratio between its network delay and the maximum network delay (nd_{max}) among all links in the hierarchical WAN of resources: $nd_{max} = \max_{i,j}\{nd_{E_iF_j}, nd_{F_jC}\}$.

As the Edge nodes are not used for processing data streams, each entire raw data stream S_j produced by an IoT area A_j at the Edge reaches the Fog anyway. Thus, the cost part concerning the Edge to Fog network links is fixed (set as c constant), we then focus on minimizing the Fog to Cloud network resource usage cost:

$$nru = c + \sum_{j=1}^{N} nbu_{F_jC} \cdot \frac{1}{nb_{F_jC}} \cdot \frac{nd_{F_jC}}{nd_{max}} \quad (3)$$

The network bandwidth effectively used on all the Fog-to-Cloud network links is defined as follows: $B = \sum_{j=1}^{N} nbu_{F_jC}$

III. Response time aware resource allocation problem

In the core model of section II, we assumed that the operators of G can be deployed only between the Fog and Cloud. According to the criteria of minimizing cru and nru, the resulting G_{dep} defined in Formula (1) becomes the disjoint partition in subgraphs $Gmig_j$ to deploy on the Fog nodes F_j and $Gmig_C$ to deploy on the Cloud node C. In this respect, each $Gmig_j$ is delimited with $Gmig_C$ by a replication and migration point, *an edge-cut* ec_j. However, G_{dep} should also take into account any real-time response constraints imposed to a DSPA application. Thus, prior to introduce the problem statement, we formalize the response time of DSPA application.

A. Response time model

We define the response time T as the worst end-to-end latency $L\pi_{ij}$ among all operator paths π_{ij} in G_{dep} [19], each data stream S_j is processed by $n_\pi > 0$ operator paths $\pi_{ij} \in G_{dep}$ with i=$\{1 \dots n_\pi\}$:

$$T = \max_{\pi_{ij} \in Gdep} (L\pi_{ij}) \quad (4)$$

To calculate the end-to-end latency $L\pi_{ij}$ of an operator path π_{ij}, we consider the network delays of each network link traversed by this operator path along with the latency of each operator $O_x \in \pi_{ij}$ for processing its data load D_x:

$$L\pi_{ij} = \max_{\forall E_i \in m_j(t)} (nd_{EiFj}) + \sum_{e_{xy} \in \pi_{ij}} nd_{\mathcal{M}(O_x),\mathcal{M}(O_y)} + \sum_{O_x \in \pi_{ij}} l_x \quad (5)$$

1) Network link delays: Given that data streams are exclusively processed between the Fog and Cloud layers, Formula (5) includes only the worst network delay nd_{EiFj} among all Edge to Fog network links. Then, $\mathcal{M}$ is the mapping function, that gives the resource node n_j=$F_j|C$ on which an operator O_x is (or can be) mapped to, and $nd_{\mathcal{M}(O_x),\mathcal{M}(O_y)}$ is the network delay for transmitting data between two resource nodes on which the operators O_x and O_y are mapped to. $nd_{\mathcal{M}(O_x),\mathcal{M}(O_y)}$ is negligible if the operators O_x and O_y are placed on the same resource node.

In general, the network delay includes the propagation delay of the network link that depends on the distance between the connected nodes, the processing and queuing delays of a packet at the intermediate routers and the transmission delay which depends on the available bandwidth on the network link [20]. By assuming that the processing delay is a small constant as the router need to analyse only the data packet header to get its destination and send it to its destination, the network delay can be formulated as the sum of the propagation delay and the transmission delay [21]: $nd_{n_jn_k} = pd_{n_jn_k} + td_{n_jn_k}$.

a) Propagation delay ($pd_{n_jn_k}$): is the time it takes to transmit bits of data between two connected nodes and it is independent on the data size [16]. However, it depends on the type of the network link medium, the distance between the connected endpoints and is limited by the speed of the light. To approximate the propagation delay between two nodes, we can use the Vivaldi algorithm [22].

b) Transmission delay ($td_{n_jn_k}$): is the time for sending data on a network link from node n_j to node n_k. It depends on the size of data to transmit and the available network bandwidth. The latter is impacted by several factors including the number of active sessions, the transmission capacity of the link (nominal network bandwidth capacity), the medium access control, etc. Several techniques have been proposed to estimate the available network bandwidth of a network link [21]. In our work, to estimate the transmission delay of any data d_{xy} of size $|d_{xy}|$ from the operator O_x mapped on the node n_j to the operator O_y mapped on the node n_k, we first evaluate the transmission delay $td'_{n_jn_k}$ of a data d of considerable size $|d|$ from node n_j to node n_k, the actual transmission delay of data d is $td'_{n_jn_k}$ = $td'_{n_jn_k}$ - $pd'_{n_jn_k}$. Then, the transmission delay of any data d_{xy} is calculated as follows [16]: $td_{n_jn_k}$= $td'_{n_jn_k} \cdot \frac{|d_{xy}|}{|d|}$

2) Operator latency: The latency of an operator O_x depends on its current data load D_x, the type of operation it performs (e.g., filtering, projection, aggregation, etc.) and the available computational resource in terms of CPU (cpu_j) of the hosting node $n_j = F_j|C$. Thus, let μ_x be the rate at which an operator O_x can process its data load D_x on a node n_j [23]: $\mu_x = cpu_j/req_x$.

We assume that the resource nodes use a time sharing overbooking strategies to enable CPU allocation even if the CPU demand is greater than the total CPU capacity. Thus, if $MIPS_j$ is the total CPU capacity of a resource node n_j, then cpu_j=$min(MIPS_j, (MIPS_j/(q + q_j)))$. Where, q and q_j are respetively the number of current running processes and the operators to replicate on the node n_j [16].

Given the infinite nature of data stream, let wt_x be the waiting time of data elements in an operator queue. The service rate μ_x, the waiting time wt_x and the number of data elements in an operator queue are random variables over a continuous time parameter. It is not so easy to calculate the operator latency l_x. Thus, we model each operator as a queuing system with one server and following the first in first out policy [24]. Then, the operator latency l_x is approximated as follows: $E(l_x) = E(wt_x) + \frac{1}{\mu_x}$. To approximate wt_x, we need to consider the characteristic of each operator O_x in G, in particular, whether it relies on count based or time based windows.

Determining the characteristic of each operator enables to describe each operator queuing model with the Kendall's notation in the form of X/Y/1 where X is the data arrival rate distribution, Y the service rate distribution and 1 server [24]. Consequently, to formulate the associated waiting time $E(wt_x)$.

In this paper, we assume that the operators implement time based sliding window. This window consider the invariable temporal extend of the window called window time ω_x, the progression temporal step, called sliding time β_x where $\omega_x > \beta_x$ [25]. The window contains the set of data that arrives within the last ω_x time units, so that window data are processed every β_x time units. The size of each window D_x is dynamic and dependents on the actual IoT data stream rate. The data arrival rate λ_x to an operator O_x may follow an exponential distribution [25]. As the service rate depends on the size of the data to process, it also follows an exponential distribution. Hence, we model a time based sliding window operator as a M/M/1 queuing system where the approximated wt_x is: $E(wt_x) = \frac{\lambda_x}{\mu_x \cdot (\mu_x - \lambda_x)}$.

B. Optimization Problem

Given the overall (Fog) computational resources usage cost, i.e., cru and the overall (Fog-to-Cloud) network resource usage cost, i.e., nru, the optimal placement $\mathcal{M}$ of $Gdep$, between the Fog and Cloud layers should minimize both cru and nru while satisfying the resource usage and the real-time constraints. As simultaneously minimizing nru and cru is conflicting, we consider a single objective optimization of the overall resource usage cost RU defined as the weighted sum of the normalized forms of cru and nru, respectively to CRU and to NRU by applying the min-max scaling technique:

$$\text{minimize} \quad RU = w_c \cdot CRU + w_n \cdot NRU \tag{6}$$

$$\text{subject to} \quad cmu_{F_j} \leq cm_{F_j}\ j = \{1, \ldots, N\}, \tag{7}$$

$$B \leq Bmax \tag{8}$$

$$T \leq Tmax. \tag{9}$$

Where $w_c \geq 0$ and $w_n \geq 0$ are respectively the weights for computational and network resource usage cost, which enable to specify a usage preference between the two types of resources.

Formula (8) constrains the Fog to Cloud bandwidth usage by an upper threshold $Bmax$. Due to the limited computational resources in Fog, Formula (7) constrains the usage of these resources to guarantees that the CPU/memory usage of each Fog node F_j does not exceed the available CPU/memory. To ensure the real time constraint, Formula (9) imposes that the response time of a specific DSPA application should not exceed a threshold $Tmax$

For processing a data stream, each operator $O_x \in G$ requires computational and network resources to process and transmit data stream with a certain latency (l_x). We can show that the edges in G represent the precedence constraint between operators as we model G as a DAG of operators. Furthermore, by defining T as the maximum end-to-end latency $L_{\pi_{ij}}$ among all the individual operator paths π_{ij}, we can show that the critical operator path is the path with the maximum end-to-end latency $L_{\pi_{ij}}$, with $L_{\pi_{ij}} > Tmax$. Hence, we need to deploy G (as G_{dep}) between the nodes $n_i = F_j|C$ to minimize the end-to-end latency of the critical operator path π_{ij} while ensuring fairness in resource usage [26]. The latter is defined in terms of: (i) optimal trade-off between CRU and NRU, (ii) resource usage constraints (i.e. Formulas (7) and (8)) and (iii) operator replicability constraint. This problem is an instance of the job shop scheduling (JSS) problem known to be NP-hard [26]. Its complexity increases as we increase the number of resource nodes or the size of G.

Given that computational and network resources of H can be shared by several DSPA applications, the available resources may vary in time as long these applications can be deployed and removed on the fly. Moreover, the number and the rate of IoT data streams S_j may also vary according to the mobility patterns of IoT devices [10]. Under these conditions, an optimal placement $\mathcal{M}$ statically defined may not anymore be a feasible solution to our TSOO problem. For this reason, we need to dynamically reschedule an already deployed application graph G_{dep} at run-time by identifying a new operator placement $\mathcal{M}$ that optimizes RU and satisfies the constraints.

IV. Time based Single Objective Optimization (TSOO)

To solve our TSOO problem, we need to search in G the edge-cut ec_j of each data stream S_j, in order to build the graph G_{dep} that includes each individual subgraph $Gmig_j$ to deploy on the Fog node F_j and $Gmig_C$ to deploy on the Cloud node C. Each $Gmig_j$ should satisfy the constraint $cmu_{Fj} \leq cm_{Fj}$ and the operator replicability constraint. Moreover, the processed data streams S_j that will be transmitted on the Fog to Cloud network links should jointly satisfy the constraint $B \leq Bmax$. Finally, the DSPA application response time T should satisfy the constraint $T \leq Tmax$.

We observe that when selecting an edge-cut ec_j as the replication and migration point of a data stream S_j, we can individually calculate the effect of this selection on RU. Thus, we assume that the computational and network resource usage costs can be split for each individual stream S_j: $RU = \sum_{j=1}^{N} RU_j$, where RU_j=CRU_j+NRU_j and RU_j, CRU_j and NRU_j are respectively the contributions to RU, CRU, and NRU for processing S_j.

Our solution, called TSOO Algorithm, extends the SOO algorithm [11]; we recall below the main functions that we use: (i) **RUminCut()** minimizes directly RU. To do so, it first considers $Grep$ as the subgraph of G containing the set of operators that can be replicated on the Fog, so as to ensure the operator replicability constraint. Then for each data stream S_j, it identifies the subgraph $Gmig_j \in Grep$ delimited by the edge-cut $ec_j \in Grep$ that produces the minimum RU_j while $Gmig_j$ satisfies the constraint $cmu_{Fj} \leq cm_{Fj}$. (ii) **DataMinCut()** minimizes NRU. It processes like $RUminCut()$. However, it identifies the subgraph $Gmig_j \in Grep$ based on the edge-cut $ec_j \in Grep$ that produces the minimum NRU_j.

A. Algorithm description

TSOO algorithm (cfr. Algorithm 1) starts by applying $RUminCut()$ to generate a solution that attempts to minimize directly RU. If the output solution of $RUminCut()$ satisfies the constraints $B \leq Bmax$ and $T \leq Tmax$, then the achieved RU is optimal (line 6). Otherwise the problem may not have a solution or, if a solution exists, finding the optimal solution is NP-hard.

As a next step, TSOO algorithm applies a greedy search that produces local optimal solutions to approximate the global optimal solution in a reasonable amount of time. In this respect, we apply the function $dataMinCut()$ to identify the solution that minimizes NRU and consequently B (lines 7-8). If the output solution of $dataMinCut()$ does not satisfy the constraint $B \leq Bmax$, the TSOO problem has no solution satisfying this constraint, unless we relax $Bmax$ in order to accept this solution as the least bad one. If the constraint $B \leq Bmax$ is satisfied, TSOO further checks whether the constraint $T \leq Tmax$ is satisfied or not: (i) if the constraint $T \leq Tmax$ is satisfied, this means that $dataMinCut()$ produces a good solution satisfying all the problem constraints; (ii) if the constraint $T \leq Tmax$ is not satisfied, starting from the solution of $dataMinCut()$, we search in the subgraphs $Gmig_j$ and $Gmig_C$ the

operators to move from the Fog to the Cloud (or the inverse) in order to reduce T until we satisfy the constraint (lines 10-13). This greedy search is performed with the functions $operatorMoveback()$ and $operatorMoveDown()$, described in Section IV-B. If the greedy search produces a solution where the constraint is still not satisfied, the TSOO problem has no solution satisfying this constraint, unless we relax $Tmax$ to accept this last solution as the least bad one.

If both constraints $B \leq Bmax$ and $T \leq Tmax$ are finally satisfied, TSOO applies a greedy search to minimize RU while keeping satisfied the problem constraints (lines 14-31).

Algorithm 1: TSOO

Input: G, application graph
Input: Grep, subgraph of replicable operators of G
Input: $Bmax$, upper threshold for bandwidth usage
Input: $Tmax$, upper threshold for response time
Input: Sraw, set of raw data streams S_j

1 $X \leftarrow \emptyset$, set of S_j on which ΔRU is applied S_j
2 $M \leftarrow \emptyset$, set of $Gmig_j$, replicable subgraph per S_j
3 $RM \leftarrow \emptyset$ set of edge-cuts $ec_j \in Grep$ per S_j
4 $RU \leftarrow 0$, overall resource usage cost
5 $B \leftarrow 0$, Fog-to-Cloud network bandwidth usage
6 $M \leftarrow RUminCut()$
7 **if** $B > Bmax$ **then**
8 $M, RM, G_{dep} \leftarrow dataMinCut(G)$
9 **if** $T > Tmax \wedge B \leq Bmax$ **then**
10 Set $max\pi$, sorted set of paths π_{ij} where $L\pi_{ij} > Tmax$
11 $operatorMoveBack(M, RM, G_{dep}, max\pi)$
12 **if** $T > Tmax$ **then**
13 $operatorMoveDown(M, RM, G_{dep}, max\pi)$
14 **if** $B \leq Bmax \wedge T \leq Tmax$ **then**
15 ΔRUset $\leftarrow edgeCutMove(G, RM, M)$
16 Sort ΔRUset in increasing order
17 Pull ΔRU on top of ΔRUset
18 **while** $\Delta RU < 0$ **do**
19 Get $S_j, e_{j_k}, Gmig_{j_k}$ corresponding to Δ_{RU}
20 $ec_j \leftarrow RM[j]$
21 Calculate T when considering ec_j
22 $B^{'} \leftarrow B - |ec_j| + |ec_{j_k}|$
23 **if** $B^{'} \leq Bmax \wedge T \leq Tmax \wedge X.has(S_j)$ = *False* **then**
24 $RM[j] \leftarrow e_{j_p}$
25 $M[j] \leftarrow Gmig_{j_k}$
26 $B \leftarrow B - |ec_j| + |ec_{j_k}|$
27 $RU \leftarrow RU + \Delta_{RU}$
28 $X.add(S_j)$
29 Pull ΔRU on top of ΔRUset
30 Rewrite G_{dep} to include all $Gmig_{nj}$ (or $Gmig_{j_p}$) $\in$ M
31 Send each $Gmig_{nj} \in M$ to corresponding resource node n_j

B. Satisfy the Response time constraint

When moving an operator from the Cloud to the Fog, the operator latency will probably increase, assuming that the computational resources of a Fog node are smaller than the ones of the Cloud. At the same time, if other operators of the same operator path are already hosted on this Fog node, the latency of these operators will also increase, as the node resources will now be shared among more processes. Regarding the operators of the same path that remain in the Cloud after this move, we assume that the effect on their latency is negligible. On the other hand, the effect of this move on the network link delay between the Fog node and the Cloud depends on the size of the data produced by the moved operator in comparison to the size of the data that were transmitted from the Fog node to the Cloud before. In the opposite case, when moving an operator from the Fog to the Cloud, the operators latency on the same path will probably decrease, while the network link delay may increase or decrease.

In this respect, to satisfy the response time constraint we first apply the function $operatorMoveback()$, as it is more likely to reduce the response time T on an operator path. More specifically, we iteratively select the operator paths π_{ij}, where $L\pi_{ij} > Tmax$, in decreasing order of their end-to-end latencies $L\pi_{ij}$. For each selected π_{ij}, we start from the edge-cut ec_j that delimits the two subgraphs $Gmig_j$ and $Gmig_C$, through which this operator path π_{ij} traverses, and we select the upstream replicated operator of ec_j to be removed from the Fog node F_j. If this action improves the resulting end-to-end latency, we continue to remove the next upstream replicated operator, as long as the constraint $B \leq Bmax$ is satisfied. We stop applying $operatorMoveback()$ if the constraint $T \leq Tmax$ is satisfied. However, if the constraint is finally not satisfied or if removing a replicated operator does not improve the resulting end-to-end latency, we next apply the function $operatorMoveDown()$.

The $operatorMoveDown()$ processes similarly. Rather than removing the replicated operators from the Fog, it replicates the not yet replicated operators on the Fog. Then, it stops if the constraint $T \leq Tmax$ is satisfied. Otherwise, the TSOO problem does not have a solution that satisfies the real-time response constraint. Due to space limitations, we omit the pseudo code of the two functions.

C. Run-time monitoring for rescheduling operator placement

DSPA engines such as Apache Flink [27] run several DSPA applications. It consists of Job Manager (JM) and multiple Task Managers (TMs) distributed across the nodes. By assuming that data streams are processed on Fog and Cloud nodes, the JM can be deployed on Cloud as it provides practically unlimited resources and a TM can be deployed on each Fog and Cloud nodes [5]. The JM is responsible for planning the execution of operators by the mean of a scheduler and to assign them to the TMs. Each TM execute the assigned operators and it uses a local monitor to continuously monitors the operator metrics i.e. data arrival rate (λ_x) and service rate (μ_x), the computational resource usage (cmu_{n_i}) and the data stream size (nbu_{F_jC}) to sent to the Cloud. Then it reports periodically these average monitored values to the JM for diagnosis. JM uses a global monitor to aggregate the reported values and to calculate: RU, B and T. and it decides whether $\mathcal{M}$ is still an acceptable solution under the following conditions:

$$RU \leq RUmax, \tag{10}$$

$$\sum_{k=1}^{k_{max}} \gamma_k = 0 \tag{11}$$

Where RU_{max} is an upper threshold of RU defined by the application owner and γ_k a term that penalizes the violation of any constraint k of the TSOO problem under the following conditions:

$$\gamma_k = max(0, \frac{val_k - max_k}{max_k}) \tag{12}$$

For each constraint k, val_k can be the value of cmu_{Fj}, B or T and max_k can be the value of cm_{F_j}, $Bmax$ or T_{max}.

If at least one of the conditions (10) and (11) is not satisfied, JM triggers the TSOO algorithm for rescheduling the operator placement. The current monitoring mechanism may incur overhead due to operator migration. Thus, existing mechanism for autonomous system such as Monitor, Analyze, Plan and Execute (MAPE) loop pattern can be used for this purpose [28].

V. Performance evaluation

We use iFogSim to simulate an Edge-Fog-Cloud architecture as well as several DSPA applications. Besides the TSOO algorithm, our experimental framework also implements 3 competitor methods.

TSOO-SA: uses the simulated annealing (SA) meta-heuristic [15] to solve our TSOO problem. Given that SA solves unconstrained optimization problems, we use a constraint relaxation method [15]: we add to the cost function to minimize (i.e., RU) a penalty ζ_k for the violation of each individual constraint k of the TSOO problem:

$$f = w_c \cdot CRU + w_n \cdot NRU + \sum_{k=1}^{k_{max}} \zeta_k \quad (13)$$

We set ζ_k = 1 if $\gamma_k > 0$ (see Formula (12)), otherwise ζ_k = 0.

SA starts from an initial solution or current solution f_i and, based on an adequate perturbation method, generates a neighbour solution f_j of the current one. SA accepts a neighbour solution to be the new current one on the basis of a probability controlled by a parameter called temperature τ. SA considers also: (i) the cooling rate α, which determines how quickly the temperature decreases; (ii) I, number of transitions for each value of τ, which determines after how many neighbour solutions generated, SA will decrease the temperature; and (iii) τ_{final}, which determines the temperature at which SA will end. To set these parameters for TSOO-SA, we use the values proposed in [15]: τ=300, τ_{final}=10, α=0.9 and I=100 per each value of τ.

To generate a neighbour solution, we use the mutation technique Shift Move (SM) [29]. In the current solution, we first represent as a key value table the mapping of each stream S_j and its replication and migration point ec_j. Then, from this table, we randomly select a data stream S_j and we replace its current replication and migration point ec_j by ec'_j. ec'_j is the next edge-cut of ec_j in the source-to-sink direction of G. If ec_j includes a sink of G, ec'_j will take the value of the first edge-cut in G (source-to-sink direction). Then, we move to right for all the streams $S_k \neq S_j$, the replication and migration points from one position.

TRCS (Time and Resource Constraint Satisfaction): extends the baseline approach RCS [14]. TRCS enhances pure IoT Cloud analytics by dynamically placing the operators between the Fog and the Cloud in synergy with the evolution of the IoT data stream rates. More specifically, assuming an initial deployment of all the operators in the Cloud, TRCS minimally uses the Fog computational resources to satisfy the constraints $T \leq Tmax$, $B \leq Bmax$ and $B \geq Bmin$, where $Bmin$ is a lower threshold of B used to avoid the oscillation of operator placement between the Fog and the Cloud.

Rizou et al [16]: propose an algorithm to distribute operators on a peer resource network. In this respect, the algorithm considers a latency (i.e., network delay) space where each host has a virtual position. It determines for each operator its optimal hosting node that minimises its network resource usage in the latency space, depending on the data stream rates of its neighbour operators. Starting from this solution, if the response time constraint is not satisfied, the algorithm moves each operator to the candidate host that increases the network usage as little as possible until satisfying the response time constraint.

A. Environment and system setup

1) ***Dynamic IoT data streams:*** We statically simulate the variability of the data stream rates arriving to the Fog nodes by selecting randomly (uniform distribution) 10 values of M IoT devices in the interval $[5000, 50000]$, where each IoT device produces data at a rate of 4KB/s [30]. Then, for each value of M, we set an interval $[0, M]$ in which we uniformly set $m_j(t)$ IoT devices per geographical area A_j, so that the sum of $m_j(t)$ be equal to M. In this respect, each Fog node F_j receives data stream S_j at rate $|S_j| = m_j(t) \times 4KB/s$. We wish to have 15 results of T, B, RU, and CRU for each value of M and to plot the average of these results per value of M. Hence, we repeat the splitting of each value of M per geographical area 15 times. In this respect, the total data rate reaching the Fog follows a sequence of 10 uniformly distributed values of M $\times 4KB/s$ repeated 15 times. We feed this sequence to TSOO and the baselines.

2) ***Fixed Edge-Fog-Cloud architecture:*** The topology of the hierarchical Edge-Fog-Cloud architecture [11] includes 1 Cloud node at the top of the hierarchy, 10 Fog nodes in the middle layer, and up to 50000 IoT devices at the Edge at the bottom. We use the tool Ether [22] to generate plausible network configurations of the Edge-Fog-Cloud architecture. The distribution of the resulting network configurations follows the one used in [5]. The propagation delay and network bandwidth for each Edge to Fog network links are respectively in the interval $pd_{E_iFj} \in [10, 100]$ ms and $nb_{EFj} \in [10, 50]$ Mbps and for each Fog to Cloud network links they are respectively in the interval $pd_{F_jC} \in [100, 300]$ ms and $nb_{FjC} \in [100, 250]$ Mbps. For the computational resources, we simulate the Cloud node as an Amazon VM instance of type m6g.xlarge with 12GB of memory and 35900 MIPS of CPU [6]. At the Fog, we simulate ESXi virtual machines where the CPU capacities are set between 2400 and 8150 MIPS and the memory capacities are set between 1 and 4 GB [31]. The MIPS evaluation comes from [32].

3) ***DSPA application:*** We consider a model of the TLC application (New York City Taxi and Limousine Commission rides) [33]. The TLC application finds the busiest driver every two hours, where each vehicle emits at the end of a ride a data record containing driver identification, pick-up and drop-off times and locations. It comprises 5 operators and 1 sink (see Figure 2), where each operator uses a time based window and is modeled as a M/M/1 queue.

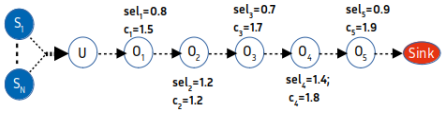


Fig. 2: DSPA application used in the evaluation

4) ***Setting parameters:*** We set Bmax=125MB/s to avoid overloading the Fog to Cloud bandwidth usage, as in [11] and for TRCS we set $Bmin$=30MB/s. Setting RUmax too high leads to high resource usage, which may lead to the system becoming unstable, while setting RUmax too low leads to inconsistent optimization of RU. Thus, we set $RUmax = 0.9$. Given that the maximum propagation delay among all the network links is 300ms, we set $Tmax$=500ms.

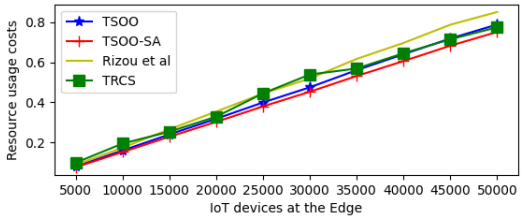
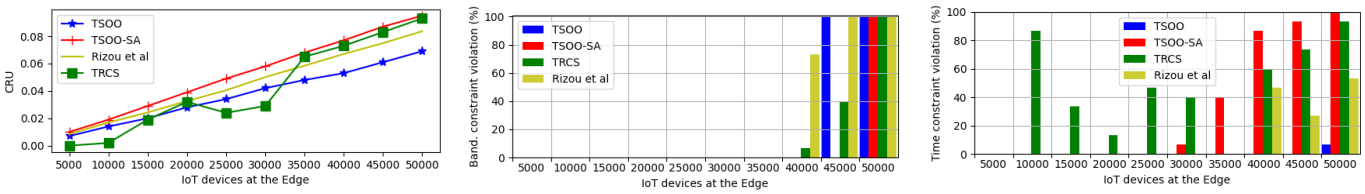


Fig. 3: Overall resource usage cost

B. Evaluation results

1) ***Resource usage cost:*** Figure 3 shows that TSOO approximates the optimal RU with an approximation error of up to 5.34%, when comparing to TSOO-SA. Furthermore, TSOO outperforms Rizou et al. and TRCS, whatever the number of IoT devices (and the produced data stream rates), with a difference ratio respectively up to 11.32%



(a) Fog computational resource usage cost (b) Violation rate of bandwidth constraint (c) Violation rate of response time constraint

Fig. 4: Computational resource usage cost and constraint violation rates

and 23.23%. Figure 4a shows that TSOO has the lowest CRU when comparing to TSOO-SA and Rizou et al. Moreover it increases with a practically constant value to offset the increasing of the data stream rate at each M value. Even though that TRCS has the lowest CRU for certain value of M, as it is expected to use less Fog computational resources. However it does not optimize CRU.

*2) **Constraint satisfaction**:* As we need absolutely to have a solution to deploy for each value of M, we consider constraint relaxation when an algorithm does not find a solution that satisfies all the constraints. Therefore, we analyze whether an achieved RU satisfies or not our problem constraints. To do so, we divide by 15 the number of times, the value of B (or T) fails to satisfy the related constraint. This gives the constraint violation rates, and shows how likely an algorithm can satisfy the constraints per each value of M.

Cloud bandwidth usage constraint ($B \leq Bmax$). As depicted in Figure 4b, TSOO satisfies this constraint whatever the data stream rates for up to M=40000. However, TSOO does not satisfy this constraint at higher data stream rates ($M \geq 45000$); it results in 100% of constraint violations. TSOO-SA fails to satisfy this constraint only for the highest data stream rate (M=50000 IoT devices), also with 100% of constraint violations. As for TRCS and Rizou et al., they do not satisfy this constraint starting from $M \geq 40000$, with less than 100% of constraint violations.

Response time constraint ($T \leq Tmax$). Figure 4c shows that TSOO satisfies this constraint whatever the data stream rates, except for the highest data rate ($M = 50000$), where it may fail to fulfill this constraint with 6.6% of constraint violations. TSOO-SA satisfies this constraint only at lower data stream rates ($M \leq 25000$). Upon moderate or higher data stream rates, it may not satisfy this constraint with an increasing percentage of constraint violations (i.e. 6.67% to 100%) as the data stream rates increase. TRCS is likely to fail to satisfy this constraint with up to 93.3% of constraint violations upon $M > 5000$. This is due to the fact that TRCS prioritizes the satisfaction of the constraint $Bmin \leq B \leq Bmax$.

Finally, Rizou et al. has 0% of constraint violations for $M < 40000$, while it starts failing to satisfy this constraint for $M \geq 40000$. However, the constraint violation rate is between 26.67% and 53.3%. This is attributed to the priority of the algorithm in satisfying the constraint $T \leq Tmax$ after identifying the minimum NRU.

*3) **Scalability analysis**:* We assess the scalability of TSOO against TSOO-SA and Rizou et al. in terms of execution cost, i.e., the time it takes for each algorithm to find the best operator placement, to rewrite the application graph based on this placement, and to deploy the resulting application graph. In this respect, we consider 5 different DSPA applications built based on the TLC application. Thus, App-1 has 5 operators (cf. Figure 2), App-2 has 6 operators, ..., and App-5 has 9 operators. We keep the same configuration of the Edge-Fog-Cloud architecture as presented in Section V-A2 and V-A4. We plot only the results for M=35000 IoT devices.

Figure 5 (A) shows that RU is practically constant for each one of the algorithms when identifying the operator placement of App-1, App-2 and App-3 with respectively 5, 6 and 7 operators. TSOO-SA produces the lowest RU among the three algorithms, and TSOO outperforms Rizou et al. As a matter of fact, the operator placement that each algorithm identifies for App-1 is the same for App-2 and App-3, at least regarding the resources that are taken into account in RU. On the other hand, TSOO-SA does not produce the lowest RU for the operator placement of App-4 and App-5, which have higher numbers of operators (8 and 9). This is due to the fact that the current operator placement, which is considered at run-time as the initial solution of TSOO-SA, may not be close to the optimal solution (as it is required [15]). Thus, during the iterations for improving this solution, TSOO-SA fails to converge towards the optimal solution.

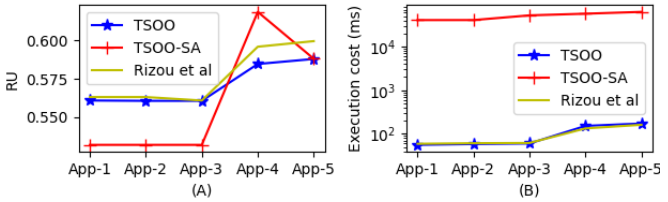


Fig. 5: Resource usage cost (A) and Execution cost (B).

In Figure 5(B), the execution cost is increasing with the number of operators per DSPA application. TSOO-SA has the highest execution cost, with a difference ratio of up to 729% when comparing to TSOO. TSOO's execution cost is very close to the one of Rizou et al. with a difference ratio of up to 0.12% (TSOO being slightly more costly).

VI. Related Work

Dynamic scheduling of DSPA applications is largely discussed in the context of resource rich execution environments(i.e. Cloud, cluster, etc.) than in the context of Edge/Fog computing due to the lack of solid established edge-oriented DSPA engines [34], that why we use simulation tool to evaluate our proposed scheduling solution. In the context of resource-rich execution environments, Lohrmann et al. [20] distribute DPSA application in cluster nodes with the objective to minimize resource usage cost and keep the response time under a certain threshold. Arkian et al [35] propose a MAPE loop pattern for dynamic scheduling operators only at the Fog layer which providing high computational resources aiming to maximize the throughput. Heinze et al. [36] focus on dynamic scheduling of DSPA applications in distributed homogeneous hosts. Then, their objective is to maximize the resource usage while keeping the response time under a certain threshold by using an operator migration technique. In our work, we are interested in continuously scheduling DSPA computations between heterogeneous Fog and Cloud nodes where the Fog nodes come with limited computational resources. Our objective is to satisfy both the response time and bandwidth usage constraints. In this respect, we are searching for an optimal trade-off between the Fog computational resource usage cost and the Fog to Cloud network resource usage cost.

Nowadays, machine learning (ML) techniques are used to recognize patterns from measured or profiled data to dynamically

schedule operators in the Edge-Fog-Cloud continuum. For instance, reinforcement learning (RL) is used in [37] for dynamic scheduling that minimizes the DSPA application response time. The problem is modeled as a Markov Decision process (MDP) that explores two RL techniques (Q-learn and Monte-Carlo-Search-Tree). Furthermore, [38] studies dynamic operator scheduling on heterogeneous infrastructure with the goal of minimizing the response time, resource usage cost and reconfiguration overhead. MDP model suffers from the dimensionality of the problem to model and hence the state space increases as the problem size increases that may bear high execution cost when comparing to a heuristic approach.

Meta-heuristic are largely used as ways of obtaining better solutions for NP-hard problems. For instance, SA is used in [39] to minimize the response time and energy consumption when placing application modules between the Cloud and the Edge. Particle swarm optimization is used by [40] to efficiently place application modules in the Edge-Fog-cloud continuum by minimizing energy consumption and response time while respecting the constraints of resource node capacities and module placement. The challenge in this context is to identify good setting parameters and a best scheme to generate neighboring solutions that may converge to an optimal solution. These approaches come with a high execution cost to be used for a dynamic operator placement.

VII. Conclusion

In this paper, we addressed the problem of dynamically scheduling operators between Fog and Cloud nodes, in response to the dynamic data stream rates produced at the Edge. We proposed TSOO, a resource-aware and time-efficient algorithm that takes into account the limited Fog computational resources, the real-time response constraints, and congestion and delay issues on Fog-to-Cloud network resources. Experimental results showed that TSOO is scalable and time efficient. It approximates the optimal solution by managing the trade-off between the usage costs of Fog computational resources and the Fog-to-Cloud network resources and satisfies the real-time constraint. As future work, we plan to extend TSOO to take as input the current operator placement from which a minimum set of changes will be applied in case of rescheduling rather than to run from scratch the whole algorithm. Furthermore, we plan to also include computational resources of the Edge layer.

References

[1] A. Tiwari, B. Ramprasad, Mortazavi *et al.*, "Reconfigurable streaming for the mobile edge," in *Proceedings of the 20th HotMobile*, 2019.

[2] P. Dangal and G. Bloom, "Towards industrial security through real-time analytics," in *ISORC*. IEEE, 2020.

[3] H. Röger *et al.*, "A comprehensive survey on parallelization and elasticity in stream processing," *ACM CSUR*, 2019.

[4] R. Deng, R. Lu, C. Lai, T. H. Luan, and H. Liang, "Optimal workload allocation in fog-cloud computing toward balanced delay and power consumption," *IEEE internet of things journal*, vol. 3, no. 6, 2016.

[5] A. Jonathan, A. Chandra, and J. Weissman, "Wasp: wide-area adaptive stream processing," in *the 21st Middleware Conference*, 2020.

[6] Amazon, "Aws whitepapers guides," https://amzn.to/3dJ0rLL, 2021, [Online; accessed 28-July-2021].

[7] L. Bittencourt, R. Immich, R. Sakellariou, N. Fonseca, E. Madeira *et al.*, "The internet of things, fog and cloud continuum: Integration and challenges," *Internet of Things*, 2018.

[8] M. Iorga, L. Feldman, R. Barton, M. J. Martin, N. S. Goren, C. Mahmoudi *et al.*, "Fog computing conceptual model," 2018.

[9] A. Ali-Eldin, B. Wang, and P. Shenoy, "The hidden cost of the edge: A performance comparison of edge and cloud latencies," 2021.

[10] Y.-W. Hung, Y.-C. Chen, C. Lo, A. G. So, and S.-C. Chang, "Dynamic workload allocation for edge computing," *VLSI*, 2021.

[11] P. Ntumba, N. Georgantas, and V. Christophides, "Efficient scheduling of streaming operators for iot edge analytics," in *FMEC 2021-Sixth International Conference on Fog and Mobile Edge Computing*, 2021.

[12] M. Harchol-Balter, *Performance modeling and design of computer systems: queueing theory in action*. Cambridge University Press, 2013.

[13] H. Gupta, A. Vahid Dastjerdi *et al.*, "ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments," *Software: Practice and Experience*, 2017.

[14] P. Ntumba, N. Georgantas, and V. Christophides, "Scheduling continuous operators for iot edge analytics," in *Proceedings of the 4th EdgeSys@EuroSys*, 2021.

[15] D. Delahaye, S. Chaimatanan *et al.*, "Simulated annealing: From basics to applications," in *Handbook of metaheuristics*. Springer, 2019.

[16] S. Rizou, F. Diirr, and K. Rothermel, "Fulfilling end-to-end latency constraints in large-scale streaming environments," in *30th IEEE International Performance Computing and Communications Conference*. IEEE, 2011.

[17] L. Li *et al.*, "Online workload allocation via fog-fog-cloud cooperation to reduce iot task service delay," *Sensors*, 2019.

[18] T. Djemai, P. Stolf, T. Monteil, and J.-M. Pierson, "Mobility support for energy and qos aware iot services placement in the fog," in *2020 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*. IEEE, 2020.

[19] V. Cardellini, V. Grassi, F. Lo Presti, and M. Nardelli, "Optimal operator placement for distributed stream processing applications," in *10th ACM DEBS*, 2016.

[20] B. Lohrmann, P. Janacik, and O. Kao, "Elastic stream processing with latency guarantees," in *2015 IEEE 35th International Conference on Distributed Computing Systems*. IEEE, 2015.

[21] R. Prasad, C. Dovrolis, M. Murray, and K. Claffy, "Bandwidth estimation: metrics, measurement techniques, and tools," *IEEE network*, 2003.

[22] T. Rausch, C. Lachner, P. A. Frangoudis, P. Raith, and S. Dustdar, "Synthesizing plausible infrastructure configurations for evaluating edge computing systems," in *3rd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 20)*, 2020.

[23] E. G. Renart, A. D. S. Veith, D. Balouek-Thomert *et al.*, "Distributed operator placement for iot data analytics across edge and cloud resources," in *19th IEEE/ACM CCGRID*, 2019.

[24] J. F. Shortle, J. M. Thompson, D. Gross, and C. M. Harris, *Fundamentals of queueing theory*. John Wiley & Sons, 2018, vol. 399.

[25] Q. Jiang and S. Chakravarthy, "Queueing analysis of relational operators for continuous data streams," in *Proceedings of the twelfth international conference on Information and knowledge management*, 2003.

[26] V. A. Strusevich, "Shop scheduling problems under precedence constraints," *Annals of operations research*, vol. 69, 1997.

[27] P. Carbone, A. Katsifodimos *et al.*, "Apache flink: Stream and batch processing in a single engine," *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, vol. 36, no. 4, 2015.

[28] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, 2003.

[29] H. Hosseini Nasab and F. Mobasheri, "A simulated annealing heuristic for the facility location problem," *International Journal of Mathematical Modelling and Numerical Optimisation*, vol. 4, no. 3, 2013.

[30] F. Xhafa, B. Kilic, and P. Krause, "Evaluation of iot stream processing at edge computing layer for semantic data enrichment," *Future Generation Computer Systems*, 2020.

[31] C. Wöbker, A. Seitz, H. Mueller, and B. Bruegge, "Fogernetes: Deployment and management of fog computing applications," in *Network Operations and Management Symposium*. IEEE, 2018.

[32] "7-Zip LZMA Benchmark." [Online]. Available: https://www.7-cpu.com/

[33] P. Silva, A. Costan *et al.*, "Investigating edge vs. cloud computing tradeoffs for stream processing," in *IEEE Big Data*, 2019.

[34] V. Cardellini, F. Lo Presti *et al.*, "Run-time adaptation of data stream processing systems: The state of the art," *ACM CSUR*, 2022.

[35] H. Arkian, G. Pierre, J. Tordsson *et al.*, "Model-based stream processing auto-scaling in geo-distributed environments," in *30th ICCCN*, 2021.

[36] T. Heinze, Z. Jerzak, G. Hackenbroich, and C. Fetzer, "Latency-aware elastic scaling for distributed data stream processing systems," in *the 8th ACM International Conference on Distributed Event-Based Systems*, 2014.

[37] A. da Silva Veith, M. D. de Assunçao, and L. Lefevre, "Monte-carlo tree search and reinforcement learning for reconfiguring data stream processing on edge computing," in *2019 31st SBAC-PAD*. IEEE.

[38] G. R. Russo, V. Cardellini, and F. L. Presti, "Reinforcement learning based policies for elastic stream processing on heterogeneous resources," in *Proceedings of the 13th ACM DEBS*, 2019.

[39] J. Fang, K. Li, J. Hu, X. Xu, Z. Teng, and W. Xiang, "Sap: An iot application module placement strategy based on simulated annealing algorithm in edge-cloud computing," *Journal of Sensors*, vol. 2021, 2021.

[40] T. Djemai, P. Stolf, T. Monteil, and J.-M. Pierson, "A discrete particle swarm optimization approach for energy-efficient iot services placement over fog infrastructures," in *2019 18th ISPDC*. IEEE, 2019.