Adaptive Scheduling of Continuous Operators for IoT Edge analytics

Patient Ntumba, Nikolaos Georgantas, Vassilis Christophides

Adaptive Scheduling of Continuous Operators for IoT Edge analytics

Patient Ntumba[a,*], Nikolaos Georgantas[b], Vassilis Christophides[c]

[a] *Cnam Paris, France*
[b] *MiMove Team, Inria Paris, France*
[c] *ENSEA, ETIS, France*

Abstract

In this paper, we address the problem of adaptive scheduling of data stream processing and analytics (DSPA) applications in a shared edge fog cloud (EFC) continuum with response time constraints. The focus is on handling the dynamic workload of DSPA applications caused by the variability of their input data stream rates generated by mobile IoT devices, and the dynamically available resource capacity in the EFC continuum. To address these challenges, we characterise the different types of resources in the EFC continuum, as well as the operators that make up a DSPA application. Based on this characterisation, we propose models to evaluate the response time and the cost of using the resources in the always dynamic EFC continuum. We then formulate the problem of adaptive scheduling of a DSPA application in the EFC continuum with the objective of minimising the cost of using the shared resources subject to the constraints of the response time and the available capacity of the EFC resources. We propose a heuristic algorithm that dynamically computes a new scheduling of the DSPA application, taking into account its current deployment state and the current state of the shared resources in the EFC continuum. Experimental results, using simulation, show the effectiveness of our proposed algorithm against algorithms of related work.

Keywords: IoT, Data stream, Continuous operator, Edge/Fog, Cloud, Optimization, Queueing theory, Dynamic system

1. Introduction

The Internet of Things (IoT) is a prominent technology that connects an increasing number of physical devices that produce large amounts of data. Following this trend, several use cases are emerging that collect and process these volumes of IoT data in real time to support many application domains such as urban mobility, smart cities, healthcare, augmented reality, etc.

In particular, we consider the class of applications known as *Data Stream Processing and Analytics (DSPA)* applications, which are used to process unbounded data streams in order to extract valuable information in a timely manner using a series of *continuous operators* such as aggregation, filtering, joining, etc. [1]. The data stream processing engine such as Apache Flink [2], Apache Kafka [3], Nebullastream [4], etc., are the main frameworks used to implement and deploy the DSPA applications and were essentially designed to be deployed in the cloud [5].

Motivating example. Figure 1 shows a DSPA application that analyses data streams (e.g., vehicle ID, GPS location, driving speed) generated by connected vehicles. The DSPA application consists of 4 operators, namely operator U, which merges the raw traffic data streams into a single stream for further processing; operator J, which matches the GPS location with the in-memory road network database to find the road segment and area related to the vehicle location; operator G_{by}, which aggregates the input data stream by road segment and calculates the average speed and number of vehicles per road segment; and operator F, which filters the results of the previous operator to obtain the overall traffic status by area, which is fed to the sink node for Countrywide Traffic Monitoring (CTM). This application is intended to provide the traffic monitoring status in a timely manner, i.e., within the response time constraint, to reflect the actual traffic conditions on the roads. When deploying this application in the cloud, it is assumed that the traffic data is transmitted from the IoT devices to the cloud. With high variability in traffic density, the bandwidth of the wide area network (WAN) connecting to the cloud can become a limiting factor. This is because the network can experience significant delays due to the high volume of data streams and the changing Internet traffic conditions over time [6].

Figure 1: DSPA App. for Countrywide Traffic Monitoring (CTM)

In this work, we adopt the Edge-Fog-Cloud (EFC) computing paradigm as a solution for deploying the DSPA applications. Specifically, EFC computing enables local processing of IoT data near edge/fog nodes located at the edge of the IoT network. This approach allows data to be processed closer to the IoT devices before being sent to the cloud. Therefore, it effectively reduces network latency and congestion issues in the Cloud network compared to the traditional Cloud-based deployment paradigm[7]. As shown in

*Corresponding author at: Cnam, Paris, France.
Email address: patient.ntumba@cnam.fr (Patient Ntumba)

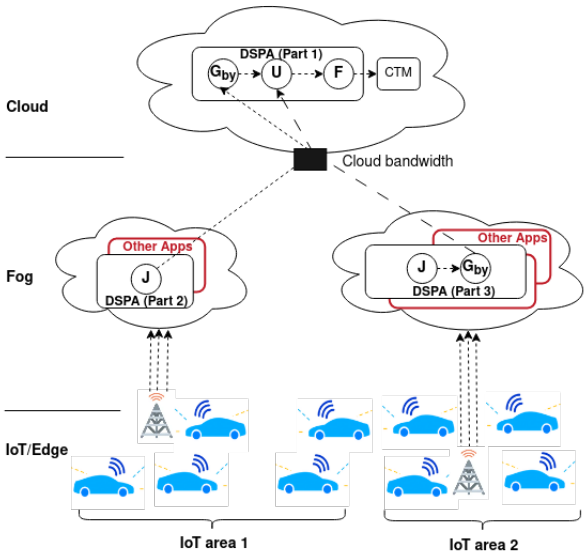


Figure 2: DSPA application splits between Fog and Cloud for processing IoT data streams

Figure 2, we distribute the DSPA application for country-wide traffic monitoring between the EFC nodes. Each part of the DSPA application is deployed at individual fog nodes, providing local processing capacity to connected vehicles within the IoT area covered by each specific fog node.

Challenges of Edge-Fog-Cloud paradigm. The EFC computing comes with the following challenges to overcome [8, 9]:

- Heterogeneous computational resource nodes (e.g., street antennas, base stations, small data centers, routers, etc.) and heterogeneous network resources: wan links with varying delays and bandwidth capacities;
- Fluctuation of IoT application workload due to the spatio-temporal dynamics of mobile IoT devices at the Edge;
- Dynamic availability of capacities of computational and network resource capacity in the EFC. This is due to the fact that applications may be deployed across EFC resources or release allocated resources at different time scales, directly affecting the overall available resource capacities.

Problem statement. High workload demands of IoT application can lead to increased use of computational resources and network bandwidth, potentially resulting in increased network delays and processing latency, which can drastically affect the application time constraint [9]. Given this problem, a dynamic scheduling mechanism of the DSPA application within the EFC continuum becomes essential. The key questions are twofold: (i) How to dynamically split the DSPA application between the EFC nodes with the goal to continuously ensure the efficient use of both shared and dynamic resources, while satisfying the response time constraint ? (ii) How to split the DSPA application so that the resulting DSPA application satifies the DSPA application semantic ?

In our previous work [10], we address the problem of scheduling a DSPA application on dedicated fog and cloud resources with the objective of simultaneously minimising the computational resource capacities of the fog nodes and the network resource capacities of using the WAN links to reach the cloud, while satisfying the application response time constraint. A static heuristic algorithm was proposed: it required the scheduling solution to be computed from scratch each time.

Contributions. In this paper, we go beyond existing work by formulating a new cost model for the use of EFC resources, which may be dynamic and shared, and therefore require adaptive scheduling of the DSPA application. In this respect, the main contribution of this work is as follows:

- A novel cost model is proposed to evaluate the cost of using the computational and network resources in an EFC continuum. This model takes into account the dynamic available capacity of these two resources, favors the use of resources with high relative availability and weights the request to use a resource by the resulting utilisation ratio of the resource if the request is deployed on it;
- A response time model is introduced that takes into account the constraints associated with windowing the processing of IoT data streams by the continuous operators. This helps to capture the time-sensitive nature of data stream processing.
- Formal definition of the problem as a *Time based Single Objective Optimisation* (TSOO) to partition DSPA applications between the fog and the cloud to jointly optimise the resource usage cost of the computational and network resources. The TSOO problem is constrained by the application response time and the dynamically available capacities of the computational and network resources.
- A heuristic algorithm called Adaptive TSOO Heuristic (aTSOO-H) is proposed. aTSOO-H essentially adapts a previously found scheduling solution of DSPA application operators between the fog and cloud nodes. This is achieved according to the evolution of IoT data stream rates and the available resource capacities. The goal is to continuously minimise the combined cost of computational and network resource usage and satisfies the response time constraint. We assume that the DSPA application is not processed at the edge, but only between the fog and cloud nodes.
- Simulation experiments: The proposed aTSOO-H algorithm is evaluated through extensive simulations based on real-world data sets. When compared with the baseline solution [10] and the solutions from related work [8], the results show that aTSOO-H achieves a high dynamic deployment success rate of

DSPA applications, i.e., the best trade-off between resource usage cost, response time satisfaction and lower rescheduling cost. Rescheduling cost is defined as the difference between the number of operators that a DSPA application contains before after its scheduling. In particular, aTSOO-H has 100% of success rate at lower data stream rates, and the success rate decreases with increasing IoT data stream rates up to 21%, while the related work algorithms reach 0% in the latter case.

The remainder of this paper is organised as follows: Section 2 positions our work with respect to the literature. In Section 3, we present the cost model of using the shareable resources across the EFC architecture and the response time model of DSPA application. In Section 4, we formulate the problem and in Section 5 we propose a solution to the problem. Section 6 details the experimental methodology while Section 6 presents the results. Then, we conclude this work in Section 8.

2. Related work

The problem of scheduling tasks (operators) that constitute an application across centralised or distributed computing resources has been widely discussed in the literature under different optimisation objectives and constraints. The table 1 highlights the contributions of this work with respect to the most relevant related works.

The first criterion to group related works is the execution environment of DSPA applications. IoT data streams may be processed exclusively at the network Edge in [11, 12], or in a hierarchical or a peer to peer (P2P) networks of Edge and Cloud nodes in [13, 14, 15, 16, 17, 18, 19, 20]. EFC nodes are used in a P2P fashion in [8, 21, 22, 23] or in a hybrid fashion combining P2P and hierarchical networks in [24, 25, 26]. Our work focus on a hierarchical architecture of EFC nodes that can be shared among several DSPA (or other) applications as also studied in [8, 26].

Most of the works that rely on EFC (or Edge-Cloud) architecture focus on optimising the network resource usage and the DSPA response time by constraining only the computational resource usage [19, 15, 13, 8, 26]. In our work, a cost model is introduced to deal with the trade-offs of minimising the joint usage of both computational and network resources. The model allows to characterise the usage cost of resources by distinguishing between abundant and constrained shareable resources. Few works considers in their optimisation goals the computational resources usage [21, 27] or the response time constraint [8].

Different optimisation approaches are used for operator scheduling, either using only the placement policy or both the replication and placement policies. The works in [28, 29, 30] rely on integer linear programming (ILP) to model the problem of scheduling DSPA applications in geo-distributed P2P network resources, with the aim of minimising the response time and the network resource usage and maximising the availability of DSPA applications. The optimisation objective is constrained by the capacities of the computational and network resources and solved using the optimization software tool CPLEX. The works [13, 21] exploit the Mixed Integer Linear Programming (MILP) to model the problem of scheduling operators between the edge and the cloud. The objective was to minimise the response time of the DSPA application, the monetary cost of using the computational and the network resources. The constraint was to guarantee the throughput of the DSPA application. The MILP model was also solved with CPLEX.

Scheduling based on mathematical optimisation (ILP, MILP, etc.) suffers from high execution costs and raises serious scalability concerns. For these reasons, heuristic algorithms have been proposed in the literature. For instance, [31] dynamically schedules operators across distributed fog nodes with the goal of maximising the (maximum sustainable) throughput of a DSPA application. Unlike our work, this solution neither optimises the use of computational resources nor considers a hierarchical EFC architecture. Moreover [15, 14] schedule operators between edge nodes and federated Cloud nodes, with the objective to minimise the DSPA application response time along with network resource usage and monetary cost. A heuristic method is first employed to find an initial deployment (static scheduling). Then, dynamic scheduling is modelled as a Markov Decision Process (MDP) which is solved by using reinforcement learning (RL) techniques. Unlike our work, this scheduler does not minimise the computational resource usage of the edge nodes.

The work in [8] addresses on-the-fly scheduling of multiple DSPA applications over shared EFC resources, but only statically. The aim is to maximise the number of successfully deployed DSPA applications, while limiting the use of network bandwidth for transmitting data streams over the fog-to-cloud WAN links and making efficient use of fog computing resources. The scheduling algorithms proposed in [8] have the following goals: i) to limit the use of network bandwidth for transmitting data streams over the fog-to-cloud WAN links; ii) to use the fog computing resources efficiently so that they can also serve other applications; and iii) to save the fog computing resources for DSPA applications with strict latency requirements and use the cloud for non-time-critical DSPA applications. In our work, we consider a DSPA application whose sink is located in the cloud (i.e., CTM in Figure 1). Therefore, among all the proposed scheduling algorithms in [8], only the FogOnly algorithm is relevant for deploying between the fog and the cloud DSPA applications with fixed sinks in the cloud. However, FogOnly attempts to maximise the use of fog computing resources. When applying the other scheduling algorithms of [8] to our problem, it falls that the whole DSPA application is deployed entirely in the cloud. In this case, we compare the solutions found aTSOO-H with a baseline inspired by the goals of the algorithms proposed in [8].

3. Core model

In this section, we present the system architecture of EFC computing and the DSPA application. We then model the main cost factors of running DSPA applications on EFC computing, which provides shared network and computing resources. Finally, we model the response time of DSPA applications.

Table 1: Summary of related works

	Our work	Nardelli et al[28]	Arkian et al[31]	De souza et al[13]	Da Silva et al[15]	Rzepka et al[8]
Infrastructure						
Cloud	✓	✓	×	✓	✓	✓
Fog	✓	×	✓	×	×	✓
Edge	✓	×	×	✓	✓	×
Resource network						
Hierarchical	✓	×	×	×	×	×
Shareable	✓	×	×	×	×	✓
Objective						
Min. Net. usage.	✓	✓	×	✓	✓	×
Min. Comp. usage.	✓	×	×	✓	×	×
Constraint						
Bandwidth	✓	✓	✓	✓	✓	✓
CPU/RAM	✓	✓	✓	✓	✓	✓
Response time	✓	✓	✓	×	×	✓
Replicability	✓	×	×	×	×	×
Strategy						
Heuristic	✓	×	×	×	✓	✓
Static	✓	✓	✓	✓	✓	✓
Dynamic	✓	✓	✓	×	✓	×
Policy						
Placement	✓	✓	✓	✓	✓	✓
Replication	✓	✓	✓	✓	×	×

3.1. Edge-Fog-Cloud architecture

The terms Edge and Fog computing are often used interchangeably [32]. However, in this work we consider them as different layers where the boundary between the two is tiny [33]. In particular, we consider that Edge/Fog layers provide complementary computational and network resources to the Cloud enabling IoT edge analytics in the EFC continuum as depicted in Figure 3.

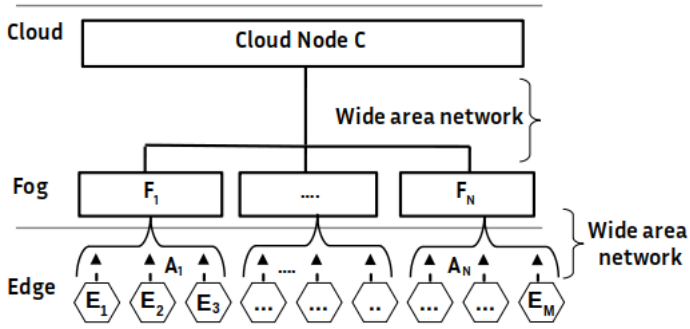


Figure 3: Hierarchical Edge-Fog-Cloud Architecture

We abstract an EFC architecture as a hierarchical wide-area resource network defined by the set H={E,F,C} [25]. 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 consists 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. In this respect, we consider S_j as the aggregation of data streams arriving to a Fog node F_j and produced by $m_j(t) \leq M$ IoT devices moving at a time t in the geographic area A_j. Given the above, the number ,M, of available IoT devices at the Edge at a time is given by Formula (1)

$$M = \sum_{j=1}^{N} m_j(t) \tag{1}$$

In this architecture, we distinguish between the computational resources of the Edge/Fog/Cloud nodes in terms of memory (i.e., RAM) for executing operators and the network resources in terms of the bandwidth and delay of each WAN link connecting two nodes through which data stream are transmitted from an operator to another operator.

3.1.1. Computational resources

For each individual Edge/Fog/Cloud node, we consider in terms of memory (i.e., RAM): (i) the maximum computational resources cm_{E_i}, cm_{F_j} and cm_C for respectively the Edge node E_i, the Fog node F_j and the Cloud node C; (ii) the available computational resource capacities at a time t cma_{E_i}, cma_{F_j} and cma_C for respectively the Edge node E_i, the Fog node F_j and the Cloud node C.

More specifically, a physical or virtual node may be dedicated to a single DSPA application. For example, a Raspberry pi at the Edge or a VM in the Cloud. In this case, cm_{E_i} is the maximum capacity of the physical node at the Edge and cm_C is the maximum reserved capacity of the VM in the Cloud. Then, the available capacity $cma_{E_i} < cm_{E_i}$ or $cma_C < cm_C$ may occur when some operators of the DSPA application are already deployed on respectively this Edge node or this Cloud node. In a different case, a physical or virtual node may be shareable among multiple DSPA applications and / or other processes. For example, a gateway server at the edge or a VM in the Fog. In this case, cm_{E_i} is the maximum capacity of the physical node at the Edge and cm_{F_j} is the maximum reserved capacity of the virtual node in the Fog, shared among multiple processes. Here, the available capacity $cma_{E_i} < cm_{E_i}$ or $cma_{F_j} < cm_{F_j}$ may occur due to the DSPA application of interest and / or due to other (DSPA) applications.

3.1.2. Network resources

Let $nb_{E_iF_j}$ be the maximum network bandwidth to reach a Fog node F_j from the closest Edge nodes E_i and nb_{F_jC} the maximum network bandwidth on the network link from a Fog node F_j to the Cloud node C. While, $nba_{E_iF_j}$ is the available network bandwidth capacity on the network links from the Edge to the nearest Fog nodes F_j and nba_{F_jC} is the available network bandwidth on the network link from the Fog node F_j to the Cloud node C.

Several techniques have been proposed to estimate the available network bandwidth of a network link. The interested reader can refer to [34]. Practically speaking, a network link is a logical link dedicated to a DSPA application, which however shares the same underlying physical links with other (DSPA) applications. In this respect, if nb_{F_jC} is the maximum bandwidth (best effort, not reserved) of the logical link from the Fog node F_j to the Cloud C that can be used by a DSPA application, the available network bandwidth $nba_{F_jC} < nb_{F_jC}$ may occur due to the utilisation of this network link by the DSPA application of interest but also due to the utilisation by other (DSPA) applications.

The network delay on a network link is the time it takes for the first byte to arrive to the destination. It depends on the distance between the source and the destination of this network link as well as on the network congestion due for example on the available network bandwidth capacity, the data size to transmit on this network link, etc. In this respect, let $nd_{E_iF_j}$ be the network delays of the network link from an Edge node E_i to its nearest Fog node F_j and nd_{F_jC} the network delay of the network link from a Fog node F_j to the Cloud node C. Table 2 gives a summary of the parameters we use to model a EFC architecture.

Table 2: List of symbols used to model the EFC architecture

Symbol	Description
H	Set of resource nodes in the EFC architecture
E_i, F_j, C	Respectively Edge node, Fog node, Cloud node
n_i,	Abstracts a resource node at Edge, Fog or Cloud
A_j	Geographical area of IoT devices
m_j	Number of IoT devices per geographical area A_j
cm_{E_i}	Max. computational resource capacity of node E_i
cm_{F_j}	Max. computational resource capacity of node F_i
cm_C	Max. computational resource capacity of node C
cma_{E_i}	Avail. computational resource capacity of node E_i
cma_{F_j}	Avail. computational resource capacity of node F_j
cma_C	Avail. computational resource capacity of node C
cmu_{E_i}	Computational resource demand on node E_i
cmu_{F_j}	Computational resource demand on node F_j
cmu_C	Computational resource demand on node C
$nb_{E_iF_j}$	Max. network bandwidth capacity from E_i to F_j
nb_{F_jC}	Max. network bandwidth capacity from F_j to C
$nba_{E_iF_j}$	Avail. network bandwidth capacity from E_i to F_j
nba_{F_jC}	Avail. network bandwidth capacity from F_j to C
$nbu_{E_iF_j}$	Network bandwidth demand from E_i to F_j
nbu_{F_jC}	Network bandwidth demand from F_j to C
$nd_{E_iF_j}$	Network delay of the link from E_i to F_j
nd_{F_jC}	Network delay of the link from F_j to C

3.2. DSPA application

DSPA application relies on the principle of online computation to mine data stream in near real time [29]. In this respect, a DSPA application is constituted of operators that process the data stream. Conversely to one shot operators of snapshot-based queries in traditional databases, the results of an operator are constantly updated each time new data tuple of input data streams are processed [5].

We represent a DSPA application as a directed acyclic graph (DAG) of operators (or simply application graph), denoted by G, with V_G as the set of vertices which represent the operators O_x (where $x \in \mathcal{N}$) and E_G as the set of edges which represent the data stream (data flowing) from a source node to an operator, between two operators (e.g., 0_1 to O_2), and from an operator and a sink node. Note that, G topology further includes the sources that produce the raw data streams S_j, ($j \in \mathcal{N}$) of rate $|S_j|$ consumed by the operators, and the sink fixed in the cloud that captures the final results. For example in Figure 4, the application graph G is constituted with the set of vertices O_1,O_2,O_3,O_4,O_5,O_6 and O_7. S_1 is the source and the sink nodes are $sink_1$ and $sink_2$. The edges are the data flowing for example from S_j to O_1, from O_1 to O_2, or from O_5 to $Sink_2$.

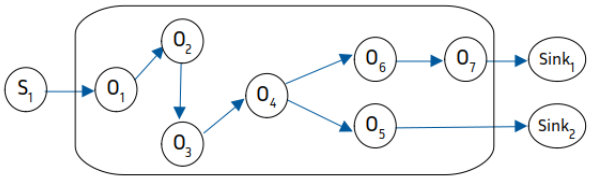


Figure 4: Graph G modelling a DSPA application

To cope with the infinite nature of data streams, we consider that operators are executed in time windows ω_x to process a finite set of data items d_x arising within a time interval. The application graph G is thus characterised by the following parameters, which are summarised in the table 3

Operator selectivity (sel_x). defined the operator selectivity as the ratio between the input and output data rate of an operator O_x.

Operator cumulative selectivity ($csel_x$). defined as the product of operator selectivity from a source to a target operator O_x according to their topological order in the application graph G. For example, in the Figure 4, the cumulative selectivity of the operator O_6 is $csel_6 = sel_1 \cdot sel_2 \cdot sel_3 \cdot sel_4 \cdot sel_6$.

Edge data rate ($\lambda_{x,y}$). defined as the rate of data flowing between two operators, from an source node x to an operator y or from an operator x to a sink node y.

Operator cost (c_x). defined in terms of memory demand (e.g.,memory per megabyte of data) for an operator O_x to process its data load D_x [35]. For example, if an operator O_1 has a cost $c_1 = 1.2$, this supposes that O_1 needs 20% of additional memory to execute its current data load.

data load (D_x). defined as the aggregation of the input data streams per time window ω_x:

$$D_x = \sum_{i=1}^{I} \omega_x \cdot \lambda_{i,x} \quad (2)$$

Where I is the number of upstream operators O_i producing data stream at rate $\lambda_{i,x}$ towards the operator O_x.

Operator resource demand (req_x). defined as the computational resource (i.e., memory) required by an operator O_x to process its data load D_x at a time t with respect to its associated cost c_x:

$$req_x = D_x \cdot c_x \quad (3)$$

For example, if an operator O_1 has $D_1 = 5MB$ and $c_1 = 1.2$, then $req_1 = 7.5MB$.

To replicate and migrate (place) a part of G on different compute resources $n_i = E_i|F_j|C \in H$, we need to partition G into disjoint subgraphs, denoted by $Gmig_i$, according to some workload criteria. Therefore, the resulting graph to be deployed is defined as follows:

$$G_{dep} = \bigcup_{\forall_{n_i \in H}} Gmig_i \quad (4)$$

To specify a replication and migration point in G, we rely on the *edge-cut* algorithm [36] which partitions G in two disjoint subgraphs. An edge-cut ec_j contains the set of edges having one endpoint in each subgraph of the partition. Additionally, let $|ec_j|$ denotes the rate of an edge-cut ec_j defined as the sum of edge data rates crossing this edge-cut.

Table 3: List of symbols used to model DSPA application

Symbol	Description
G	Application graph
O_x	Operator $O_x \in G$
e_{xy}	Edge (link) for operators O_x to O_y
S_j	Aggregation of data streams from IoT area j
$\|S_j\|$	Rate of the aggregated data stream S_j
w_x	Window size of the operator O_x
sel_x	Selectivity of the operator O_x
$csel_x$	Cumulated selectivity up to operator O_x
λ_{xy}	Data rate flowing on the edge e_{xy}
D_x	Data load of the operator O_x
c_x	Cost of the operator O_x to process its data load
req_x	Resource demand (CPU/memory) of the operator O_x
$Gmig_i$	Subgraph of G to place on a node n_i
G_{dep}	Union of subgraphs to deploy across the EFC nodes
ec_j	An edge-cut in G
$\|ec_j\|$	Data rate of the edge-cut ec_j

3.3. Usage costs of shared Edge-Fog-Cloud resources

Computational and network resources are heterogeneous throughout the EFC architecture because they can be constrained in very different degrees. Furthermore, in most cases these resources can be shared by several (DSPA) applications. It is therefore important to assess the cost of using these resources in a representative way that can ensure their most efficient use, whether for static (initial) deployment or dynamic deployment of DSPA applications.

When statically deploying a DSPA application, its current deployment state is not considered. However, since EFC resources can be shared by multiple DSPA applications, it is necessary to take into account the current state of the EFC resources on which the DSPA application is deployed. In the case of dynamic deployment, the actual states of both the DSPA application deployment and the EFC resources must also be considered. To objectively evaluates the cost of request to use a resource, we need also to take into account the state of the resource if it is selected to be used. Therefore, unlike in [37, 10, 38], we introduce a new weighting of the usage of the EFC resources based on which we assess the cost of using this resource. Table 4 summarises the parameters you use to model the resource usage cost.

3.3.1. Weighting the usage of a resource

We need to distinguish a resource node $n_i = E_i|F_j|C$ over another with their underlying network links by weighting the request of using both the computational and network resources. In this respect, for any computational or network resource, we consider:

Max. the maximum reserved resource capacity; that can be either the maximum computational resource capacity cm_{E_i}, cm_{F_j} or cm_C for respectively the Edge node E_i, Fog node F_j or Cloud node C. It can also be the maximum network bandwidth nb_{EF_j} or nb_{F_jC} on the network link respectively from Edge to Fog node F_j or the Fog node F_j to Cloud node C.

Avail. the available resource capacity; that can be either the available computational resource capacity cma_{E_i}, cma_{F_j} or cma_C for respectively the Edge node E_i, Fog node F_j or Cloud node C. It can be also the available network bandwidth capacity nba_{EF_j} or nba_{F_jC} on the network link respectively from Edge to Fog node F_j or the Fog node F_j to Cloud node C.

Req. the resource usage requested, that can be either the requested computational resource usage cmu_{E_i}, cmu_{F_j} or cmu_C for respectively deploying a subgraph $Gmig_i$ on the Edge node E_i, Fog node F_j or Cloud node C. We use Formula (5), to formally define the computational resource usage request for deploying a subgraph $Gmig_i$ on a resource node $n_i = E_i|F_j|C$.

$$cmu_{n_i} = \sum_{O_x \in Gmig_i} req_x \quad (5)$$

Where req_x defined in Formula 3 is the resource usage requested by each operators $O_x \in Gmig_i$.

Req can be also the requested network bandwidth usage nbu_{EF_j} or nbu_{F_jC} for transmitting data stream on the network link respectively from the Edge to Fog node F_j or the Fog node F_j to Cloud node C. We use Formula (6), to formally define the network bandwidth usage request for transmitting the data stream from the resource node $n_i = E_i|F_j$ to the resource node $n_j = F_j|C$ assuming that $Gmig_i$ is replicated at the edge cut ec_j and deployed on the resource node $n_i = E_i|F_j$.

$$nbu_{n_i n_j} = |ec_j| \quad (6)$$

In this respect, for a resource with the maximum resource capacity Max and available resource capacity $Avail$, the current state of this resource is the difference: $Max - Avail$. If we want to deploy a DSPA application that requires resource usage Req on this resource, the usage ratio will be as following:

$$\begin{aligned} W &= \frac{(Max - Avail + Req)}{Max} \\ &\equiv \frac{Max}{Max} - \frac{Avail}{Max} + \frac{Req}{Max} \\ &\equiv 1 - \frac{Avail}{Max} + \frac{Req}{Max} \end{aligned} \tag{7}$$

Where $W \geq 0$. The intuitive interpretation of Formula 7 is that: with the term $(1 - \frac{Avail}{Max})$, we favor using resources with high relative available capacity; and with the term $\frac{Req}{Max}$, we favor using resources with high maximum capacity.

In order to calculate the computational (or network) resource usage cost of a node n_i (or a network link) across the EFC architecture, we need to capture the fact that this resource is limited and can be shared by several other DSPA applications or processes at any time. Therefore, we need to select the resource from which the resulting W is the smallest in order to minimise the cost of using a resource in the EFC architecture.

3.3.2. Computational resource usage cost

To assess the cost of using an Edge node E_i, a Fog node F_j or a Cloud node C, we multiply the usage of each node by the dynamic weight factor defined in Formula (7). In this respect, we calculate the weights W_{E_i}, W_{F_j} and W_C of using respectively an Edge node E_i, a Fog node F_j and a Cloud node C as following:

$$W_{E_i} = 1 - \frac{cma_{E_i}}{cm_{E_i}} + \frac{cmu_{E_i}}{cm_{E_i}} \tag{8}$$

$$W_{F_j} = 1 - \frac{cma_{F_j}}{cm_{F_j}} + \frac{cmu_{F_j}}{cm_{F_j}} \tag{9}$$

$$W_C = 1 - \frac{cma_C}{cm_C} + \frac{cmu_C}{cm_C} \tag{10}$$

Given the above Formulas that weight dynamically) respectively the request of using an Edge node E_i, a Fog node F_j and a Cloud node C, the overall resource usage cost is calculated as following:

$$cru = \sum_{i=1}^{M}(cmu_{E_i} * W_{E_i}) + \sum_{j=1}^{N}(cmu_{F_j} * W_{F_j}) + (cmu_C * W_C) \tag{11}$$

Where $W_{E_i}, W_{F_j}, W_C \geq 0$, M is the total number of the Edge nodes E_i, N is the total number of the Fog nodes F_j, and C is the Cloud node.

3.3.3. Network resource usage cost

We consider that the network delays and the available network bandwidth capacities of each individual WAN links can be dynamic with regard to the network conditions[6]. In the literature [39, 29], concerning peer node networks, network delay is used as the only weight factor for differentiating network links. We additionally include network bandwidth as a weight factor: using network links of limited capacity with parsimony allows an efficient sharing among several DSPA applications.

Table 4: List of symbols used to model the resource usage cost

Symbol	Description
W_{E_i}	Weight of using computational resources on E_i
W_{F_j}	Weight of using computational resources on F_j
W_C	Weight of using computational resources on C
$W_{E_iF_j}$	Weight of using network bandwidth from E_i to F_j
W_{F_jC}	Weight of using network bandwidth from F_j to C
cru	Overall computational resource usage cost across EFC
CRU	Normalised form of cru with $cru \in$ [0,1]
nru	Overall network resource usage cost across EFC
NRU	Normalised form of nru with $nru \in$ [0,1]
RU	Overall resource usage cost across EFC

In this respect, the cost of using a network link is calculated by multiplying the requested network bandwidth usage by the weight factor of using this link and its network delay. To this end, to calculate the weight factor of using each individual Edge to Fog network link (i.e., $W_{E_iF_j}$) or each individual Fog to Cloud network link (i.e., W_{F_jC}), we use Formula 7 as following:

$$W_{E_iF_j} = 1 - \frac{nba_{E_iF_j}}{nb_{E_iF_j}} + \frac{nbu_{E_iF_j}}{nb_{E_iF_j}} \tag{12}$$

$$W_{F_jC} = 1 - \frac{nba_{F_jC}}{nb_{F_jC}} + \frac{nbu_{F_jC}}{nb_{F_jC}} \tag{13}$$

Given the weight factor, the overall network resource usage cost is formulated as following:

$$\begin{aligned} nru = &\sum_{j=1}^{N}\sum_{i=1}^{M}(nbu_{E_iF_j} \cdot W_{E_iF_j} \cdot nd_{E_iF_j}) + \\ &\sum_{j=1}^{N}(nbu_{F_jC} \cdot W_{F_jC} \cdot nd_{F_jC}) \end{aligned} \tag{14}$$

Where $W_{E_iF_j}, W_{F_jC} \geq 0$, M is the total number of the Edge nodes E_i and N is the total number of the Fog nodes F_j.

3.4. Response time model

According to the criteria of minimizing cru and nru, the resulting G_{dep} defined in Formula (4) becomes the disjoint partition in subgraphs $Gmig_i$ to deploy across the EFC architecture. However, G_{dep} should also take into account any time-constraint imposed to a DSPA application. In this respect, we need to introduce the response time model of DSPA application. Similarly to [29], we define the response time T as the worst end-to-end latency among all the end-to-end latency L_j of processing each individual data stream S_j in G_{dep}.

$$T = \max_{\forall S_j}(L_j) \tag{15}$$

Note that S_j can be processed through $n_\pi > 0$ operator paths of G_{dep}, with i={1,...,n_π}. Therefore, L_j is the worst end-to-end latency among the set of operator paths π_{ij} through which S_j is processed before reaching a sink node.

$$L_j = \max_{\forall \pi_i \in \pi_{ij}} \Big(\sum_{e_{xy} \in \pi_i} nd_{\mathcal{M}(x),\mathcal{M}(y)} + \sum_{O_x \in \pi_i} l_x \Big) \tag{16}$$

To calculate the end-to-end latency of an operator path π_i, we consider the network delay of each network link traversed by this operator path along with the latency of each operator $O_x \in \pi_i$ for processing its data load D_x. Furthermore, $\mathcal{M}$ is the mapping function, that gives the resource node n_i=$E_i|F_j|C$ on which a data source node, an operator O_x or a sink node is (or can be) mapped to. Then, $nd_{\mathcal{M}(x)\mathcal{M}(y)}$ is the network delay for transmitting data from a resource node that hosts the data source x or the operator O_x to the resource node that hosts the operator O_y or the sink y. $nd_{\mathcal{M}(x),\mathcal{M}(y)}$ is negligible if the source x or the operator O_x and the operator O_y or the sink y are placed on the same resource node (i.e., $\mathcal{M}(O_x) == \mathcal{M}(O_y)$). Otherwise it is not negligible. Furthermore, l_x is the latency of the operator O_x to process its input data load D_x.

3.4.1. Network link delay

In general, the network delay includes: (i) the propagation delay on the network link medium, which depends on the distance between the connected nodes and includes the processing and queuing delays of a packet at the intermediate routers; and (ii) the transmission delay of a packet. The transmission delay depends on the available bandwidth on the network link. Hence, the network delay can be defined as the sum of the propagation and transmission delays [34]:

$$nd_{n_i n_j} = pd_{n_i n_j} + td_{n_i n_j} \tag{17}$$

Where $pd_{n_i n_j}$ is the propagation delay between two resource nodes n_i and n_j and $td_{n_i n_j}$ is the transmission delay between these two resource nodes.

Propagation delay. The propagation delay of a network link $nl_{n_i n_j}$ is the time it takes to transmit a single bit between two resource nodes (i.e., Edge node E_i to Fog node F_j or Fog node F_j to Cloud node C); it is independent of the data size [40]. However, it depends on the type of the network link medium and the distance between the connected resource nodes; it is limited by the speed of the light. It also depends on the link conditions, e.g., network congestion. The Vivaldi algorithm is largely used in the literature to approximate propagation delays between peers in a network [41].

Transmission delay. The transmission delay is the time for putting data on the wire by the source resource node n_i in order to be transmitted on the network link $nl_{n_i n_j}$ for reaching the destination resource node n_j. It depends on the size of data to transmit and the available network bandwidth $nba_{n_i n_j}$. The latter is impacted by several factors, including the number of active sessions, the transmission capacity of the link (nominal network bandwidth capacity), the link conditions, e.g., network congestion. In this respect, in order to estimate the transmission delays, we need to know the available network bandwidth capacity. However, estimating the available network bandwidth capacity on a network link is a tedious task [34].

In this paper we proceed as follows, to estimate the transmission delay of any data d_{xy} of size $|d_{xy}|$ from the operator O_x mapped on the node n_i to the operator O_y mapped on the node n_j, where $n_i \neq n_j$, we first measure the network delay $nd'_{n_i n_j}$ of data d of considerable size $|d|$ between these two nodes. The transmission delay of data d is $td'_{n_i n_j} = nd'_{n_i n_j} - pd'_{n_i n_j}$, where $pd'_{n_i n_j}$ is the propagation delay between these two resource nodes (previously estimated) with the Vivaldi algorithm. Then, the transmission delay of any data d_{xy} is calculated as follows [40]:

$$td_{n_i n_j} = td'_{n_i n_j} \cdot \frac{|d_{xy}|}{|d|} \tag{18}$$

3.4.2. 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 resources in terms of CPU (cpu_j) of the hosting resource node. Thus, let μ_x be the rate at which an operator O_x can process its data load D_x on a resource node n_j [42] and it is formulated as following:

$$\mu_x = \frac{cpu_j}{req_x} \tag{19}$$

Where cpu_j is the available resource capacity of node n_j in terms of MIPS and req_x the computational resource demands of operator O_x in terms of MIPS (see Formula (3)).

We assume that, the resource nodes use a time sharing overbooking strategy in order to enable CPU allocation even if the CPU demand is greater than the total CPU capacity [35]. Thus, if $MIPS_j$ is the total CPU capacity of a resource node n_j, we calculate cpu_j as follows [40]:

$$cpu_j = min(MIPS_j, \frac{MIPS_j}{q_j}) \tag{20}$$

where q_j is the number of processes (including the operators) running on the resource node n_j.

Given the infinite nature of a data stream, let wt_x be the waiting time that data elements remain in the operator queue if this operator is busy. However, the service rate μ_x, the waiting time wt_x and the number of data elements in an operator queue (i.e., operator data load D_x) are random variables over a continuous time parameter. For this reason, to calculate the operator latency l_x we model each operator as a queuing system (see Figure 5) with one server and following the first in first out policy [43].

Then, by modelling each operator as a queuing system, the operator latency l_x is approximated as follows:

$$E(l_x) = E(wt_x) + \frac{1}{\mu_x} \tag{21}$$

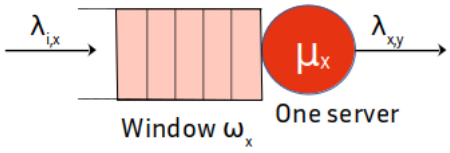


Figure 5: Modeling operator as a queuing system

To approximate the waiting time $E(wt_x)$, we need to consider the characteristics of each operator O_x in G, in particular, whether it relies on count based or time based windows.

Time-based sliding window. This type of window is characterised by the temporal extend of the window, called window time ω_x, and the progression temporal step, called sliding time β_x where $\omega_x > \beta_x$ [44]. In this respect, the window contains the set of data that arrives within the last ω_x time units, and the window data are processed every β_x time units. The data size of each window D_x is dynamic and dependent on the actual IoT data stream rate. The data arrival rate λ_x to an operator O_x may follow an exponential distribution [45].

However, given that the operator O_x always process windows of finite data D_x received at each time interval β_x, thus we can consider the arrival rate λ_x of each window is deterministic. Hence, as the service rate depends on the size of the data to process, it also follows an exponential distribution. Thus, we can model a time based sliding window operator as a D/M/1 queuing system. The waiting time is estimated as following:

$$E(wt_x) = \frac{1}{\mu_x} \cdot \frac{\gamma}{(1-\gamma)} \tag{22}$$

Where γ is the root of the equation $e^{-(\mu_x \cdot \beta_x \cdot (1-\gamma))}$ that should have the smallest absolute value. For further reading, reader can referee to [46].

Count-based window. This window considers a fixed number (K) of data to be processed. In this respect, it starts at each specified time t, selects data by going steadily backwards in time until the K data are collected. Then, the operator is triggered to process the K data contained in the window [44]. To estimate the waiting time wt_x of data element in the queue of O_x, each window is processed when K data element have arrived in the window. If the arrival rate of data follows an exponential distribution, the arrival rate of windows can be also exponential as it needs to wait until all K data items is reached. On the other hand, given that each window contains a fixed size of data to process, the service rate is deterministic. Hence, we model such an operator as an M/D/1 queuing system, where the waiting time is estimated as:

$$E(wt_x) = \frac{\rho_x}{2 \cdot \mu_x \cdot (1-\rho_x)} \tag{23}$$

Where $\rho_x = \frac{\lambda_x}{\mu_x}$ is the utilization rate of an operator O_x.

Note that, we use Table 5 to summarise the parameters we use to model the response time.

Table 5: List of symbols used for the response time model

Symbol	Description
n_i,	Abstraction of a Edge, Fog or Cloud node
$nd_{n_i n_j}$	Network delay on the network link $nl_{n_i n_j}$
$td_{n_i n_j}$	Transmission delay on the network link $nl_{n_i n_j}$
$pd_{n_i n_j}$	Propagation delay on the network link $nl_{n_i n_j}$
π_{ij}	Operator path i for processing data stream S_j
$L\pi_{ij}$	End-to-end latency of operator path
T	Response time of the DSPA application
$\mathcal{M}$	Mapping function of operators on a node
l_x	Latency of an operator O_x for processing data
μ_x	Service rate of an operator O_x
λ_x	Arrival rate of data stream to operator O_x
ρ_x	Utilisation rate of an operator O_x
wt_x	Waiting time of data in the queue of operator O_x
ω_x	Window size of of data in operator O_x
β_x	Sliding size of data in operator O_x
$MIPS_j$	Maximum CPU capacity of a resource node n_j
cpu_j	CPU resource allocated to a process of a node n_j
q_j	Total number of processes running on a node n_j

4. Problem formulation

Before formally stating the problem of dynamically scheduling DSPA applications over shared EFC resources, we assume that the resources in the cloud are practically unlimited (scalable) and can handle the evolution of the data stream rate generated by the IoT devices. However, the need to distribute the DSPA application at the edge of the IoT network stems from the fact that at high data rates, the DSPA application may experience high network delay due to the cloud network condition, which may affect the real-time constraints of the DSPA application. Furthermore, we assume that we do not consider the processing of data streams at the edge; instead, we leave this aspect for future research. Finally, we assume that the sink of the DSPA application is fixed in the cloud.

4.1. Problem statement

By assuming that the computational resources in the Cloud are practically unlimited, we consider $cm_C \to \infty$. W_C in Formula (10) can be written as $W_C = \frac{cm_C - cma_C + cmu_C}{cm_C}$. Therefore, the weight in the cloud is practically zero, $W_C \to 0$. The computational resources of Fog nodes are limited as they can not be scaled on demand, thus the weight in a Fog node is: $W_{F_j} \in]0,1]$. Finally as data stream produced by IoT devices are not processed by the Edge nodes. Then, the focus of our work is to minimise the Fog computational resource usage cost. In this respect, the overall computational resource usage cost (i.e., cru) defined in Formula (11) becomes:

$$cru = \sum_{j=1}^{N} cmu_{F_j} \cdot W_{F_j} \tag{24}$$

cmu_{F_j} is the sum of the CPU/memory usage required by each operator of the sub-graph $Gmig_j \in G_{dep}$, which is replicated on the Fog node F_j.

The Edge to the Fog network resource usage cost remains constant, regardless of how the data stream from IoT devices at the Edge reaches the Fog. We denote this fixed cost

as 'c', which represents a constant value in the overall network resource cost (referred to as 'nru') defined in Formula (14). Our main focus then shifts towards minimizing the Fog to Cloud network resource usage cost. This leads us to the modified version of Formula (14), which now becomes:

$$nru = c + \sum_{j=1}^{N} nbu_{F_jC} \cdot W_{F_jC} \cdot nd_{F_jC} \tag{25}$$

In this respect, the network bandwidth effectively used on all the Fog-to-Cloud network links is defined as follows:

$$B = \sum_{j=1}^{N} nbu_{F_jC} \tag{26}$$

While we considered Cloud computational resources are practically infinite in our resource cost model, we opt for considering Cloud network bandwidth as a resource the usage of which incurs a cost that should be taken into account. Indeed, Cloud providers rely on contracts with ISPs for network bandwidth. For distributed data intensive applications, the usage of the Cloud network bandwidth can be a bottleneck when sending huge volumes of data streams. Hence, for such applications the (monetary) cost charged by Cloud providers for network bandwidth usage can be much larger than the cost charged for computational resource usage [47]. Thus, we assume an upper threshold $Bmax$ of B which is set for a specific DSPA application.

For the response time, as we assume that the data streams are not processed at the Edge and G_{dep} is distributed only between the Fog and Cloud nodes, the operator latency at the Edge is zero and hence the end-to-end operator path latency defined in Formula (16) becomes:

$$\begin{aligned} L_j = & \max_{\forall E_i \in A_j(t)} (nd_{EiFj}) + \\ & \max_{\forall \pi_i \in \pi_{ij}} \Big(\sum_{e_{xy} \in \pi_i} nd_{\mathcal{M}(x),\mathcal{M}(y)} + \sum_{O_x \in \pi_i} l_x \Big) \end{aligned} \tag{27}$$

The first term in Formula 27 gives the maximum network delay among all the network links connecting each individual Edge nodes E_i of the IoT area A_j to the closest Fog node F_j.

We formalize the problem of minimizing nru and cru as a single-objective optimization (SOO) problem. The two metrics have different units and scales, that why we first normalize these two metrics by using the min-max scaling technique. In this respect, cru is the non-normalized form of the overall resource usage cost. Therefore, to normalize cru we devise it by the sum of the maximum capacity of all the Fog nodes F_j (i.e., cm_{F_j}). Hence, the normalized form of cru defined in Formula (24) becomes as following where CRU will take values between 0 and 1:

$$CRU = \frac{cru}{cru_{max}} \text{ where } cru_{max} = \sum_{j=1}^{N} cm_{F_j} \tag{28}$$

To normalize the overall Fog to Cloud network resource usage cost, we eliminate the constant value c, we divide the network delay of each individual Fog to Cloud network links by the maximum network delay among all the network link in order to have nru without delay unit, let named nru'. Then, we divide nru' by the sum of the maximum capacity of all the Fog to Cloud bandwidth (i.e., nb_{F_jC}) and hence, the normalized form of nru defined in Formula (25) becomes:

$$NRU = \frac{nru'}{nru_{max}} \text{ where } nru_{max} = \sum_{j=1}^{N} nb_{F_jC} \tag{29}$$

The SOO problem should take into account any response time constraint imposed by a DSPA application, we then formulate the problem as the Time based Single Objective Optimization (TSOO) problem with the aim at minimizing the overall resource usage cost RU defined as the weighted sum of CRU and NRU:

$$\begin{aligned} &\text{minimise} && RU = w_c \cdot CRU + w_n \cdot NRU && (30)\\ &\text{subject to:} && cmu_{F_j} \leq cma_{F_j}\ j = \{1,...,N\}, && (31)\\ & && B \leq Bmax && (32)\\ & && T \leq Tmax. && (33) \end{aligned}$$

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. Unlike in [38, 37] where we consider maximum computational capacity constraint, Formula (31) constrains the usage of each Fog node F_j by its available resource capacities in order to take into account the current state of the Fog node resources. Then Formula (32) constrains the Fog to Cloud bandwidth usage by the upper threshold $Bmax$ defined by the application owner. Finally the constraint (33) imposes that the response time of each DSPA application should not exceed a threshold $Tmax$, where $Tmax$ is set by the application owner.

Problem complexity. To process a data stream, each operator $O_x \in G$ requires computational and network resources to process and transmit the data stream with a certain latency (l_x). We can show that the edges in G represent the precedence constraint between operators by modelling G as a DAG of operators. Furthermore, by defining T as the maximum end-to-end latency L_j among all individual operator paths π_i, we can show that the critical operator path is the path with the maximum end-to-end latency L_j, where $L_j > Tmax$. So we need to deploy G (as G_{dep}) between nodes $n_i = F_j|C$ to minimise the end-to-end latency of the critical operator path π_i below Tmax, while ensuring fairness in resource usage [48]. Fairness in resource usage is defined in terms of: (i) optimal trade-off between CRU and NRU, (ii) resource usage constraints (i.e., Formula (31) and Formula (32)), and (iii) operator replicability constraints. This problem can be mapped onto the Job Shop Scheduling (JSS) problem, which is known to be NP-hard [48]. Its complexity increases as we increase the number of EFC nodes (the number of EFC network links) or the number of operators in the application graph G.

Dynamic scheduling problem. Given that computational and network resources of H can be shared by several DSPA (and others) applications, the resources' availability may vary in time as long as applications can be deployed and/or terminate their execution 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 [49]. Under these conditions, at run-time a (optimal) placement $\mathcal{M}$ previously calculated may not anymore be a feasible solution to the 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 $\mathcal{M}$ in order to continuously optimise RU and satisfies the problem constraints.

Such dynamic scheduling of operators requires to take into account the rescheduling cost that we define as the number of operators that are: (i) instantiated, i.e., they are replicated in the Fog or they are deployed for the first time (like union operator) in the Fog or Cloud or (ii) deleted, i.e., they are 'migrated' back to the Cloud.

5. Proposed solution

In this section, we introduce our scheduling solution called adaptive-time-based single objective optimization heuristic (aTSOO-H) algorithm. aTSOO-H algorithm encompasses the initial scheduling that deploys from scratch any DSPA application not yet deployed i.e., $\mathcal{M} == \emptyset$, (see Algorithm 1, lines 1-2). At run-time, if at least one of the TSOO problem constraints is not satisfied or RU is under/over optimised, aTSOO-H algorithm is triggered at run-time to search for a new operator placement $\mathcal{M}$ by taking into account the current operator placement $\mathcal{M}$ as $\mathcal{M} \neq \emptyset$ (see Algorithm 28 lines 3-27).

Prior to describing in detail both the initial and adaptive scheduling mechanisms of aTSOO-H algorithm, we first introduce how to partition the DSPA application between the Fog and Cloud nodes. This is to ensure that the semantic of DSPA application is guarantee at each reconfiguration of the DPSA application. aTSOO-H algorithm leverages this partitioning in order to decide which part of the DSPA application to replicate on a Fog/Cloud node or to remove from a Fog/Cloud node in order to achieve the optimisation goal with lower rescheduling cost.

5.1. Partitioning a DSPA application

To partition a DSPA application between the Fog and Cloud nodes, we split the resulting application graph G by selecting for each data stream S_j an edge-cut $ec_j \in G$ so that the sub-graph that includes S_j constitutes the sub-graph $Gmig_j$ to replicate on the Fog node F_j and the sub-graph that includes the sink, i.e., $Gmig_C$ remains in the Cloud. As defined in Formula (4), the union of all the resulting partitions of G constitutes a deployable graph G_{dep} that can be a candidate solution to our optimization goal.

For instance, Figure 6 depicts an application graph G which is entirely deployed in the Cloud to process 3 IoT data streams S_1,S_2 and S_3 and feeds the analytic results to 1 sink (Step 1). To split this application graph G between the Fog and Cloud, if we select the edge-cut $ec_3 \in G$ as the replication and migration point of S_3 and Fog node F_3, the resulting deployable graph, i.e., G_{dep} where $Gmig_3$ that contains O_2

Algorithm 1: aTSOO-H

Input: G, application graph
Input: $Bmax$, upper threshold for bandwidth usage
Input: $Tmax$, upper threshold for response time
Input: Sraw, set of raw data streams S_j
Input: $\mathcal{M}$, current operator scheduling solution
Input: RM, replication and migration points
Input: B, current overall Fog to Cloud bandwidth usage
Input: Nodes, Set of Fog node F_j where $cmu_{F_j} > cma_{F_j}$
Output: Rewrite G_{dep} to include all updated $Gmig_{n_i} \in M$
Output: Send each updated $Gmig_{n_i}$ to corresponding resource node n_i

1 **if** $\mathcal{M} = \emptyset$ **then**
2 TSOO-H(G, Grep, Bmax, Tmax, Sraw)
3 **else**
4 **for** $F_j \in Nodes$ **do**
5 Get the data stream S_j served by F_j
6 $Gmig_{current} \leftarrow M[j]$
7 Get $Gsat_j \subseteq Gmig_{current}$ where $cmu_{F_j} \leq cma_{F_j} + cmu_{current}$
8 Find minimum ec_j in $Gsat_j$
9 Find $Gmig_j \subseteq Gsat_j$ delimited by ec_j
10 $M[j] \leftarrow Gmig_j$
11 $RM[j] \leftarrow ec_j$
12 **if** $B > Bmax$ **then**
13 Set Sraw1 to contain data stream $S_j \in Sraw$ not yet migrated on Fog
14 $M, RM, G_{dep} \leftarrow$ dataMinCut(Sraw1, G)
15 Set Sraw2 $\leftarrow$ Sraw \ Sort Sraw2 in decreasing order
16 **while** $B > Bmax$ **do**
17 pull S_j on top of $Sraw2$ and get current ec_j
18 $Gmig_j, ec'_j \leftarrow dataMinCut(S_j, G)$
19 **if** $|ec'_j| \leq nba_{F_jC} + |ec_j|$ **then**
20 $M[j] \leftarrow Gmig_j$; $RM[j] \leftarrow ec'_j$
21 $B \leftarrow B - |ec_j| + |ec'_j|$
22 **if** $T > Tmax \wedge B \leq Bmax$ **then**
23 Set $max\pi$, sorted set of paths π_{ij} where $L\pi_{ij} > Tmax$
24 $operatorMoveBack(M, RM, G_{dep}, max\pi)$
25 **if** $T > Tmax$ **then**
26 $operatorMoveDown(M, RM, G_{dep}, max\pi)$
27 **if** $B \leq Bmax$ & $T \leq Tmax$ **then**
28 Improve RU

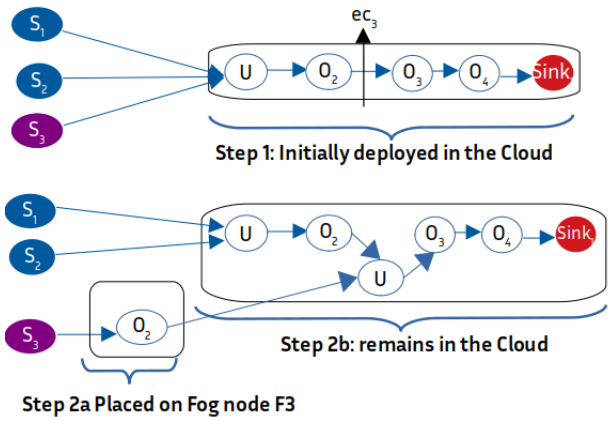


Figure 6: Partitioning a DSPA application between Fog and Cloud

Algorithm 2: operatorMoveBack

```
1  Function OperatorMoveBack(M, RM, maxπ):
2      Get π_ij on top of maxπ
3      while continue=true do
4          continue ← false
5          Get Gmig_j and Gmig_C traversed by π_ij
6          Get e_xy in ec_j of the path π_ij
7          Gmig'_j ← Gmig_j \ O_x
8          Gmig'_j ← Gmig_j \ {O_x,{e_xy}}
9          Replace e_xy by e_ux in ec_j and calculate L'π_ij, B
10         if L'π_ij < Lπ_ij & B ≤ Bmax then
11             Gmig_j ← Gmig'_j
12             Update Gmig_C accordingly and calculate T
13             M[j] ← Gmig_j, M[C] ← Gmig_C, RM[j] ← ec_j
14             if T ≤ Tmax then
15                 continue ← false
16             else if For e_ux|u=source & maxπ ≠ ∅ then
17                 Get π_ij on top of maxπ
18                 continue ← true
19         else if maxπ ≠ ∅ then
20             Get π_ij on top of maxπ
21             continue ← true
22     return M, RM, T
```

Algorithm 3: operatorMoveDown

```
1  Function OperatorMoveDown(M, RM, maxπ):
2      Get π_ij on top of maxπ
3      while continue=true do
4          continue ← false
5          ec_j ← RM[j]
6          Get Gmig_j and Gmig_C traversed by π_ij
7          Get e_xy ∈ ec_j of the path π_ij
8          Gmig'_j ← Gmig_j ∪ {O_y,{e_yz}}
9          Replace e_xy by e_yz in ec_j and calculate L'π_ij, B
10         if L'π_ij < Lπ_ij & B ≤ Bmax then
11             Gmig_j ← Gmig'_j
12             Update Gmig_C accordingly and calculate T
13             M[j] ← Gmig_j, M[C] ← Gmig_C, RM[j] ← ec_j
14             if T ≤ Tmax then
15                 continue = false
16             else if For e_xz|=sink & maxπ ≠ ∅ then
17                 Get π_ij on top of maxπ
18                 continue = true
19         else if maxπ ≠ ∅ then
20             Get π_ij on top of maxπ
21             continue = true
22     return M, T, RM
```

is replicated on the Fog node F_3 to partially process S_3 (Step 2a) while S_1 and S_2 are entirely process in the Cloud by the part of G remaining (deployed) in the Cloud (Step 2b).

We observe that by selecting an edge-cut ec_j per data stream S_j, we can split RU, CRU and NRU per S_j:

$$RU = \sum_{j=1}^{N} RU_j \tag{34}$$

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.

In this respect, we are able to search in the application graph G the edge-cuts per data stream S_j that provides the minimum NRU and the minimum RU by using respectively DataMinCut algorithm and RUminCut algorithm [37].

5.1.1. DataMinCut

The algorithm searches in G, the edge-cut ec_j per data stream S_j that minimises NRU_j. In this respect, the identified edge-cut is the minimum edge-cut [17]. To do so, for each data stream S_j, it identifies the sub-graph $Gmig_j \in G$ delimited by the minimum edge-cut $ec_j \in G$ while $Gmig_j$ satisfies the constraint $cmu_{Fj} \leq cma_{Fj}$ (see Formula (31)).

5.1.2. RUminCut

This algorithm searches in G, the edge-cut ec_j per data stream S_j that minimises RU_j. It processes like $DataMinCut()$ however, it identifies the subgraph $Gmig_j \in G$ based on the edge-cut $ec_j \in G$ that produces the minimum RU_j.

5.2. Initial scheduling algorithm

For the initial scheduling, aTSOO-H algorithm applies TSOO-H algorithm (Algorithm 1, lines 1-2) that was previously introduced in [10]. However, in this work, we apply TSOO-H algorithm in order to take into the available resource capacities rather than only the maximum resource capacities as it was previously designed.

TSOO-H algorithm starts by applying $RUminCut()$ to generate a solution that attempts to minimise directly RU. If the output solution of $RUminCut()$ satisfies the constraints $B \leq Bmax$ and $T \leq Tmax$, then the achieved RU is optimal. 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-H applies a greedy search that produces local optimal solutions to approximate the global optimal. In this respect, we apply $dataMinCut()$ to identify the solution that minimises NRU and consequently B. 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-H further checks whether the constraint $T \leq Tmax$ is satisfied or not: (i) if the constraint $T \leq Tmax$ is satisfied, this means that the solution produced by $dataMinCut()$ satisfies all the problem constraints; (ii) if the constraint $T \leq Tmax$ is not satisfied, starting from the solution of $dataMinCut()$, we search in the sub-graph $Gmig_j$ and $Gmig_C$ the operators to move from the Fog to the Cloud (or the inverse) in order to reduce T until we satisfy this constraint, while keeping satisfied the constraint $B \leq Bmax$. We briefly describe

this greedy search in Section 5.2.1. If the greedy search does not find a solution that satisfies the time constraint, we relax $Tmax$ to accept this last solution as the least bad one we can find.

If both constraints $B \leq Bmax$ and $T \leq Tmax$ are finally satisfied, as a next step, TSOO-H applies a greedy search to minimise RU while keeping satisfied the problem constraints. We briefly describe the greedy search in Section 5.2.2.

5.2.1. Satisfy the response time constraint

When moving an operator from the Cloud to the Fog, the operator latency will probably increase, if we assume 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 probably increase, if we assume that 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 delay of the network link between the Fog node F_j and the Cloud depends on the size of the data produced by the moved operator in comparison to the size of the data that was transmitted from the Fog node F_j to the Cloud before. In the opposite case, when moving an operator from the Fog to the Cloud, the operator latency on the same path will probably decrease, while the network link delay may increase or decrease.

Operator move back. is firstly applied as it is more likely to reduce the response time T on an operator path. As depicted in Algorithm 2, 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 operator move back 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 operator move down.

Operator move down. described in Algorithm 3. Unlike in Operator move back algorithm, this algorithm replicates and migrates non yet replicated operators from the Cloud to the Fog. Then, it stops if the constraint $T \leq Tmax$ is satisfied. If the constraint is not satisfied or if replicating an operator on the Fog does not improve the resulting end-to-end latency, we stop improving the response time.

5.2.2. Improve the overall resource usage cost (RU)

For each individual data stream S_j, any backward move of an edge-cut ec_j in $Gmig_j$ will decrease CRU_j (consequently also CRU) while it will increase NRU_j (consequently also NRU), let consider ΔCRU and ΔNRU the change of respectively CRU and NRU. Therefore, RU may decrease or increase, so let us consider ΔRU= ΔCRU+ΔNRU this change.

In this respect, we need to identify all possible backward moves that will further decrease RU [37]. To do so, we identify all possible backward edge-cut moves, we put the identified edge-cut moves and their ΔRU values in the set ΔRUset. To apply the edge-cut moves from the set ΔRUset as the new replication points, we sort the set ΔRUset in increasing order. Then, we pull from the top of the set the smallest ΔRU. If ΔRU< 0, we apply its corresponding edge-cut move ec_{j_k} to the data stream S_j, only if the constraints $B \leq Bmax$ and $T \leq Tmax$ are satisfied. Then, we pull from ΔRUset the next lowest ΔRU to continue improving RU, as long as the remaining ΔRUset is not empty or we do not yet encounter a ΔRU ≥ 0. TSOO-H updates at most once each data stream S_j with its best (lowest) ΔRU.

5.3. Scheduling algorithm at run-time

At the run-time of a DSPA application deployed between the Fog and Cloud nodes, upon aTSOO-H algorithm is triggered, calculating its operator placement from scratch is costly in terms of execution cost (for both execution time and rescheduling cost), that why we consider to adapt the current operator placement from which we calculate a new operator placement that solves the TSOO problem (see Algorithm 1, lines 3-27) while satisfying Formula (35).

In this respect, aTSOO-H algorithm checks whether there is a Fog node F_j where the computational resource usage constraint is not satisfied. In this case, aTSOO-H selects another edge-cut as the replication and migration point. This is used to identify a new sub-graph $Gmig_j$ so that the resulting computational resource usage satisfies the constraint ($cmu_{F_j} \leq cma_{F_j}$) (Algorithm 1, lines 4-10).

In the next step, aTSOO-H checks whether the overall Cloud bandwidth usage constraint is not satisfied. In this case, aTSOO-H migrates the data stream S_j on the Fog if they are not yet processed there. To this end, aTSOO-H applies the function $dataMinCut()$ (Algorithm 1, lines 12-13). Then, aTSOO-H builds the set $Sraw2$ that contains the data stream S_j that has been already migrated in the Fog and sorts them in decreasing order by their rates (Algorithm 1, line 14-15). If the Cloud bandwidth usage constraint ($B \leq Bmax$) is still not satisfied, aTSOO-H selects on the top of $Sraw2$ the highest-rate data stream S_j then applies the function $dataMinCut()$ in order to identify a new replication and migration point with the minimum edge-cut ec_j if it exists to further reduce the overall Cloud bandwidth usage and hence, to satisfy the Cloud bandwidth usage constraint. aTSOO-H continuous to iterate by pulling the highest-rate data stream $S_j \in Sraw2$ as long as the constraint $B \leq Bmax$ is not satisfied (Algorithm 1, lines 16-20).

Then, if the constraint $T \leq Tmax$ is not satisfied, aTSOO-H searches in sub-graphs $Gmig_j$ and $Gmig_C$ the operators to move from the Fog to the Cloud (or the inverse) in order to reduce T until the constraint $T \leq Tmax$ gets satisfied. This greedy search is performed with the functions $operatorMoveback()$ and $operatorMoveDown()$

(Algorithm 1, lines 22-25), which we describe in Section 5.2.1. If both constraints $B \leq Bmax$ and $T \leq Tmax$ are finally satisfied, aTSOO-H algorithm applies a greedy search to further minimise RU while keeping satisfied the problem constraints (Algorithm 1, lines 26-27). This greedy search is described in Section 5.2.2.

It is worth noting that at the end of the algorithm execution if aTSOO-H produces a solution where at least one constraint is still not satisfied, as our strategy is to always deploy a solution, we consider constraint relaxation to accept this last solution as the least bad one.

The current deployment of DSPA application (i.e., $\mathcal{M}$) is acceptable under the following conditions:

$$RU(t) \leq RUmax, \text{ and } \sum_{k=1}^{k_{max}} \phi_k = 0 \tag{35}$$

If at least one of the conditions (35) is not satisfied, aTSOO-H algorithm is triggered aiming to search for a new operator placement $\mathcal{M}$ by taking into account the current operator placement $\mathcal{M}$ as $\mathcal{M} \neq \emptyset$.

5.4. Time complexity analysis of aTSOO-H

To solve the TSOO problem, aTSOO-H performs the searches in the application graph for all the N data streams S_j. Therefore, the time complexity depends on the size of the application graph G, i.e. $|V_G|$ and $|E_G|$ and the number of data streams S_j or Fog node F_j, i.e., N. We know that, the time complexity of the search in the application graph G is $O(|V_G| + |E_G|)$. We recall that for the application graph G, V_G is the set of vertices (i.e., operators) and E_G is the set of edges (i.e. data flowing between two operators).

For the initial deployment of a DSPA application, aTSOO-H in the worst case will apply the following 5 functions for all the N data stream S_j, these functions are based on graph search: RUmintCut, DataMinCut, OperatorMoveDown, OperatorMoveBack. In this respect, the time complexity of aTSOO-H for the initial deployment of a DSPA application is $O(5 \cdot N \cdot (|V_G| + |E_G|))$.

For the dynamic deployment of a DSPA application, aTSOO-H in the worst case first applies for the N data streams S_j the search of minimum edge-cut of the data stream S_j not yet migrated on the Fog, then the search of the new minimum edge-cut for the data stream S_j that was already migrated in the Fog. For these two searches the aim is to satisfy the Cloud bandwidth usage constraint. The time complexity is $O(N \cdot (|V_G| + |E_G|))$. Then, aTSOO-H applies for all the N data streams S_j operatorMoveBack and operatorMoveDown in order to satisfy the response time constraint. The overall time complexity becomes $O(3 \cdot N \cdot (|V_G| + |E_G|))$. Finally aTSOO-H applies the greedy search to improve RU, this search brings the overall time complexity to $O(4 \cdot N \cdot (|V_G| + |E_G|))$.

We can note that, at dynamic deployment, aTSOO-H does not apply RUmintCut. In this respect, aTSOO-H is faster than TSOO-H at the at dynamic deployment. In both cases, the time complexity is linear.

6. Experimental methodology

This section describes the baselines and the experimental setup we use to evaluate the proposed solution aTSOO-H. We use iFogSim to simulate DSPA applications sharing the same EFC architecture for processing the IoT data streams.

6.1. Baselines

TRCS. stands for time and resource constraint satisfaction, it was introduced in [10] to enhance the 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.

FogOnly. policy aims to maximize the Fog resource usage [8] by fully deploying in the Fog a DSPA application even if it has the sink in the Cloud.

*Fog Cloud Interplay (**FCInterplay**).* policy is based on the on the goals provided in the work [8]. It processes as follows: i) Send directly a data stream S_j to be processed in the Cloud if the resulting end-to-end operator path latency can not satisfy in any way the response time constraint. This is to avoid wasting the Fog computational resources. However if only the Cloud bandwidth usage constraint is satisfied; ii) a data stream S_j for which the resulting end-to-end operator path latency can meet the response time constraint without being partially processed in the Fog is placed in the Cloud to avoid wasting Fog resources. However if only it also satisfies the Cloud bandwidth usage constraint. Otherwise, it should be partially processed on the Fog. For the latter case, we identify the sub-graph $Gmig_j$ delimited by the minimum edge-cut ec_j to be replicated on the corresponding Fog node F_j; iii) For the remaining data streams S_j, for which the resulting end-to-end operator path latency can meet the response time constraint by using the Fog resources, we identify the sub-graph $Gmig_j$ delimited by the minimum edge-cut ec_j to be replicated on the corresponding Fog node F_j.

In order to evaluate all the scheduling algorithms on a common basis, we set the experiments to trigger the algorithms aTSOO-H, TRCS, FogOnly and FCInterplay at each change in data stream rates. In this respect, we evaluate also aTSOO-H against TSOO-H algorithm that calculates a new operator placement of a DSPA application from scratch at each scheduling request.

6.2. Experimental Setup

We present the parameters that we applied to simulate the EFC architecture in iFogSim along with the DSPA applications that share the resources of this architecture. We complement with the description of dynamic generation of data stream rates produced by IoT devices and the threshold parameters used in the algorithms. Since we are using simulation to evaluate our approach, we are essentially using the real-world values suggested in the literature for the experimental setup.

6.2.1. Simulated Edge-Fog-Cloud architecture

To simulate the hierarchical EFC architecture, we consider 1 Cloud node at the top, 10 Fog nodes in the middle of the hierarchy. In the bottom, we consider up to 75000 IoT devices at the Edge in order to vary the size of the data streams to be processed by the two DSPA applications. We use Ether [50] to generate plausible (based on real data set) network configurations of a EFC architecture. The distribution of the resulting network configurations follows the one used in [6]. For the computational resources, we simulate the Cloud node as an AWS VM instance of type m6g.xlarge [47]. At the Fog, we simulate ESXi virtual machines [51]. The MIPS evaluation of each resource node comes from [52]. Table 6 presents the configuration of the EFC architecture.

Table 6: Network and computational resource parameters

	Edge	**Fog**	**Cloud**
Number of nodes	[5000, 75000]	10	1
CPU (MIPS)	-	[1, 4]	35900
RAM (GB)	-	[2400, 8150]	12
Delay to up layer (ms)	[10, 100]	[100, 300]	-
Band. to up layer (Mbps)	[10, 50]	[100, 250]	-

6.2.2. DSPA applications

We consider the New York City Taxi and Limousine Commission rides use case (TLC) [53]. In this context, we build two DSPA applications: App-1 and App-2, containing respectively 9 and 11 continuous operators. We need that the two applications come with different requirement in terms of resource usage. To this end, for App-1 depicted in Figure 7a, we set the selectivity and cost of the operators respectively from [0.4,1] and [1.0,1.8], and, for App-2 depicted in Figure 7b, we set the selectivity and cost of operators respectively from [0.8,1.2] and [1.0,1.9].

App-1 is initially deployed across the EFC architecture, then follows App-2 when App-1 is processing data stream. Thus, both App-1 and App-2 are sharing the EFC resources. For each DSPA application, we set the threshold $Bmax$=125MB/s. Given that the maximum propagation delay among all the network links is 300ms (Table 6), we set the threshold $Tmax = 1000ms$, to allow some margin for operator latency and transmission delay. Finally, for TRCS we set $Bmin$=50MB/s.

6.2.3. Dynamic IoT data stream rates

We statically simulate the variability of the data stream rates arriving to the Fog nodes by selecting randomly (uniform distribution) 11 values of M (number of IoT devices) in the interval [5000,75000], where each IoT device produces data at a rate of 6KB/s. 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.

As we want to deploy both App-1 and App-2 across the EFC resources, some IoT devices at the Edge should produce data streams to be processed by App-1 while some others IoT devices should produce data streams to be processed by App-2. In this respect, we split each $m_j(t)$ between $m_{j_1}(t)$ and $m_{j_2}(t)$ so that each Fog node F_j receives data stream S_j splits between S_{j_1} and S_{j_2} with rate respectively $|S_{j_1}| = m_{j_1}(t) \times 6KB/s$ and $|S_{j_2}| = m_{j_2}(t) \times 6KB/s$ as the input of respectively App-1 and App-2.

Specifically, we want to have close but not equal rates for the input data streams of App-1 and App-2. In this respect, $m_{j_1}(t)$ has 55% of $m_j(t)$ IoT devices per geographical area A_j and $m_{j_2}(t)$ has the remaining 45% of $m_j(t)$ IoT devices. Figure 7c depicts the total input data stream rate per DSPA application for each value of M.

We need to consider around 15 repetitions for each experiment [37, 38, 10]. In this respect, we repeat the splitting of each value of M per geographical area and per DSPA application 15 times. The total data rate reaching the Fog follows a sequence of 11 uniformly distributed values of M $\times 6KB/s$ repeated 15 times. We feed this sequence to TSOO-H, aTSOO-H, TRCS, FogOnly and FCInterplay. We produce 15 results of T, B, and RU for each value of M and we plot the average of these results per value of M and per DSPA application (App-1 and App-2).

7. Experimental results

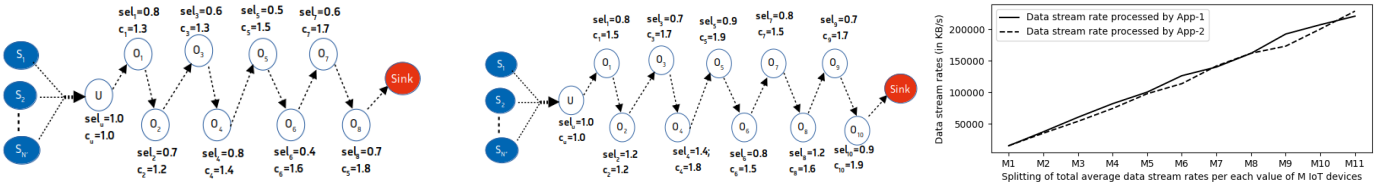
This section presents the analysis of the results obtained from the experiments conducted to evaluate aTSOO-H against baseline heuristic algorithms (TSOO-H and TRCS) as presented in Section 7.1, and against heuristic algorithms inspired by related work (FCInterplay, FogOnly) as presented in Section 7.2.

7.1. Evaluation results against baseline approaches

We compare the results achieved by aTSOO-H , TSOO-H and TRCS. We pay particular attention to the achieved results in terms of the: overall resource usage cost (RU), the satisfaction of the problem constraints related to the response time T and the Cloud bandwidth usage B and the algorithm execution cost. Figures 8, 9, 10 and 11 present the achieved results.

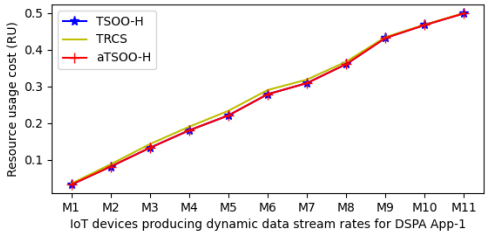
7.1.1. Overall resource usage cost

The plots of the overall resource usage cost (RU) are steeper as depicted in Figure 8a. We observe that when scheduling App-1, aTSOO-H performs like TSOO-H. In particular TSOO-H produces the optimal solution from $M1$ to $M10$ IoT devices, thanks to RUminCut algorithm, and it approximates the optimal solution at the highest data stream rates (i.e., $M11$). Since we run TSOO-H from scratch at each time the rescheduling is triggered, it is more obvious to observe such performance of TSOO-H as long as we have shown in [37] that TSOO-H is more likely to identify the optimal (or near optimal) solution. Unlike TSOO-H, aTSOO-H uses TSOO-H only for the initial deployment (i.e., first data stream rates in the random sequence of data stream rates) where it produces also the optimal solution. For the dynamic scheduling, aTSOO-H adapts the current operator placement solution in order to provide an (near) optimal operator placement solution with respect to the actual workload and available resource capacities. In this respect, aTSOO-H takes the advantage of the abundant available resources capacities across

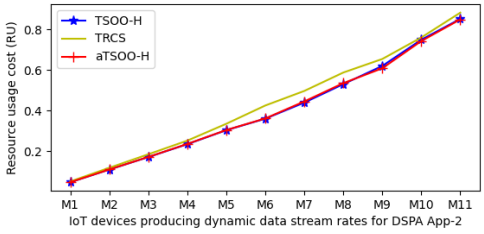


(a) DSPA application App-1 (b) DSPA application App-2 (c) Input data stream rates per DSPA apps

Figure 7: DSPA applications App-1 and App-2 sharing the EFC network resources with their input data stream rates



(a) RU of APP-1



(b) RU of APP-2 w.r.t the load of App-1

Figure 8: Overall resource usage cost per application

the EFC architecture which does not trigger constraint violations and hence it improves only RU. However TRCS provides the highest RU when comparing to both TSOO-H and aTSOO-H with a difference ratio of up to 7.65%

When Scheduling App-2 with respect to the load of App-1 already scheduled between the Fog and Cloud nodes, Figure 8b shows that TSOO-H has the lowest RU among all algorithms for the data stream rates produced from $M1$ to $M8$ IoT devices. Even though that aTSOO-H is outperformed by TSOO-H, the difference ratio is very small up to 1.29%. However, at higher data stream rates from $M9$ to $M11$ IoT devices, aTSOO-H outperforms TSOO-H with a small difference ratio of up to 1.88%. This happens in general when both algorithms fail to satisfy the constraint $T \leq Tmax$. In particular, both algorithms provide a different operator placement solution from which to improve the response time in order to satisfy the related constraint. Then, the greedy search to satisfy the response time is applied differently based on the input operator placement. Moreover, when comparing to TRCS, aTSOO-H and TSOO-H provide lower RU with a difference ratio respectively of up to 17.68% and 18.02%.

7.1.2. Constraint satisfaction rates

Cloud bandwidth usage constraint. Even though that TSOO-H and aTSOO-H provide the lowest RU when scheduling App-1, however they are more likely to fail to satisfy the constraint $B \leq Bmax$ as the data stream rates are increasing from M8 to M11 spanning from 20% to 86.67% of constraint violation rate, see Figure 9a. For App-2, we observe that TRCS is likely to not satisfy the constraint $B \leq Bmax$. As depicted in Figure 9b, both aTSOO-H, TSOO-H and TRCS start failing to satisfy this constraint at higher data stream rates (i.e., $M9$ to $M11$) with however high constraint violation rate spanning from 20% to 73.33% for TRCS, while TSOO-H and aTSOO-H have the same constraint violation rate spanning from 13.3% to 60%.

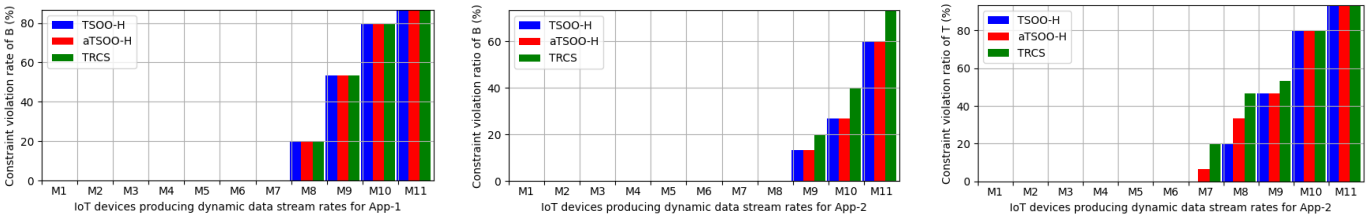
Response time constraint. All the algorithms satisfy the constraint $T \leq Tmax$ when scheduling App-1 at any data stream rates that why we omit to put the related plots. When scheduling App-2, Figure 9c shows that all the algorithms start failing to satisfy this constraint up on moderate data stream rate, i.e., M7 for aTSOO-H and TRCS and M8 for TSOO-H. Then, the constraint violation rate keep increase with the data stream rates.

7.1.3. Algorithm execution cost

We split the execution cost of the algorithm between the algorithm execution time and rescheduling cost. The execution time is the time it takes for each algorithm to find the new operator placement and to rewrite the application graph based on the identified operator placement. We define the rescheduling cost as the number of operators that are: (i) instantiated, i.e., they are replicated in the Fog or they are deployed for the first time (like operator U) in the Fog or Cloud or (ii) deleted, i.e., they are 'migrated' back to the Cloud. Improvement of the resource usage cost model will be necessary in order to take into account state migration between the Fog and Cloud in case of statefull operator. In this work we assume stateless operators.

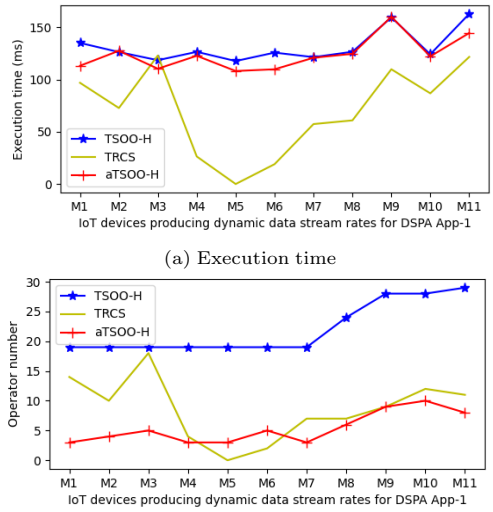
Execution time. When (re)scheduling App-1, Figure 10a shows that TSOO-H has the highest execution times from the lowest to the highest data stream rates (i.e., M1 to M11) except at M3 and M9 where aTSOO-H equals the high execution time of TSOO-H. Even thought that TSOO-H is executing in the best case from $M1$ to $M10$ by applying only RUminCut algorithm. However the adaptive approach of aTSOO-H is much faster than the RUminCut algorithm. On the other hand, we observe that TRCS has the lowest execution times among all the algorithms except at $M3$ where aTSOO-H as the lowest execution time.

For the execution time of (re)scheduling App-2 after App-1, in this case the available resource capacities are not abundant as already some part of the resources are already allocated to App-1. Therefore, the algorithms are likely to be executed in the worst case exhibiting high



(a) Cloud bandwidth constraint for APP-1 (b) Cloud bandwidth constraint for APP-2 (c) Response time constraint for App-2

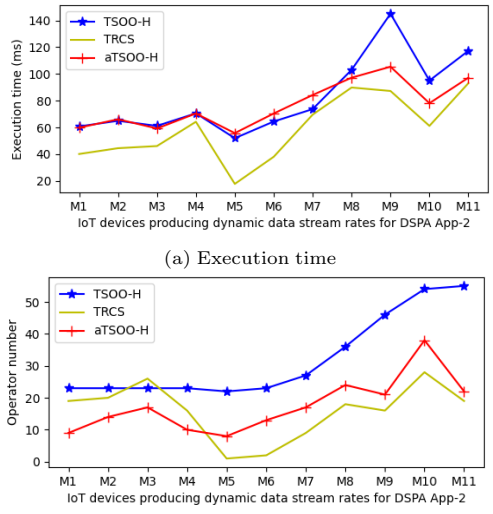
Figure 9: Constraint violation rates when scheduling App-1 and App-2



(a) Execution time

(b) Rescheduling cost

Figure 10: Execution cost of App-1



(a) Execution time

(b) Rescheduling cost

Figure 11: Execution cost of App-2

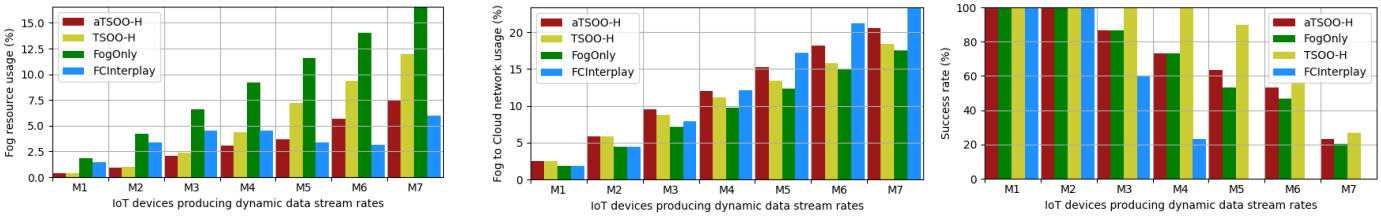
execution time. TSOO-H and aTSOO-H are still high when comparing to TRCS (see Figure 11a). However both TSOO-H and aTSOO-H have practically equal execution time at lower data stream rates i.e., $M1$ to $M4$, at moderate data stream rates i.e., $M5$ to $M7$ we can see that aTSOO-H has the highest execution time among all the algorithms. On the other hand TSOO-H has the highest execution time from $M8$ to $M11$ (high data stream rates). In this case TSOO-H is applied in the worst case that encompassed RUminCut, dataMinCut and the greedy search.

Rescheduling cost. In terms of the rescheduling cost, Figure 10b shows that the rescheduling cost of aTSOO-H is the lowest among all the algorithms except for $M5$ and $M6$. Thanks to the adaptive approach of aTSOO-H that replicates or/and removes only the operator that are likely to solve the TSOO problem. Note that TRCS outperforms aTSOO-H at data stream rates $M5$ and $M6$, this is due to fact that that TRCS only aims to keep the network bandwidth usage between $Bmin$ and $Bmax$ and therefore, TRCS moves down on Fog respectively 1 operator and 3 operators which are sufficient to satisfy the constraint $Bmin \leq B \geq Bmax$ and $T \leq Tmax$. In contrast TSOO-H has the highest rescheduling cost, this cost is constant from lower to moderate data stream rates (i.e., $M1$ to $M7$) as TSOO-H is executed from scratch in its best case by applying only the RUminCut algorithm that provides the same number of operator to replicate on the Fog. However, from moderate to higher data stream rates (i.e., M8 to M9), this cost is increasing with the data stream rate. This is due to the fact that TSOO-H is executed in the worst case where the algorithm applies a greedy search and identifies an operator placement with increasing number of operator as the data stream rates are increasing.

The rescheduling cost of App-2 after the deployment of App-1 follows the same pattern of the rescheduling cost of App-1, aTSOO-H has the lowest cost only at lower data stream rates (i.e., $M1$ to $M4$). From $M4$ to $M11$ the rescheduling cost of aTSOO-H becomes higher than the one of TRCS but remains lower than the one of TSOO-H (see Figure 11b).

7.2. Evaluation results against state of the art approaches

In this set of experiments, we wanted to evaluate aTSOO-H against the stat of the art algorithms FogOnly and FCInterplay as well as against TSOO-H. We use the same experimental setting parameters introduced earlier. However, we use the same VM in the Cloud as in the Fog. In this way the Fog can have a time advantage over the Cloud as there is no network delay. To make the evaluation fair, we consider the metrics used in [8] for the FogOnly and FCInterplay algorithms, which we can easily calculate from our model: (i) the overall Fog to Cloud bandwidth usage ratio; (ii) the overall Fog computational resource usage ratio; and (iii) the DSPA application deployment



(a) Fog computational resource usage ratio (b) Fog to Cloud bandwidth usage ratio (c) Deployment success rate

Figure 12: Computational and network utilization ratio and deployment success rate

success rate. The latter is calculated as the ratio of the DSPA applications deployed successfully over the total number of deployed DSPA applications.

In particular, we consider each change in the data stream rate as the request of deploying a new DSPA application [8]. Given the sequence of 165 variations of data stream rates that we feed to the algorithms for each of the two DSPA applications. We have in total 330 requests of DSPA application deployment with 30 requests for each of the 11 (M) values of IoT devices. Hence, the deployment success rate is calculated on the basis of 30 requests for each M value of IoT devices. In the following we present the evaluation results depicted in Figure 12. However we discuss the result for only up to M7 data stream rates, for the omitted data stream rates ($M8$ to $M11$), the deployment success rate of FCInterplay is equal to 0 as it is already the case from $M6$ and $M7$.

7.2.1. Fog computational resource usage ratio

Figure 12a shows that FogOnly has the highest usage of these resources. This is due to its strategy of deploying the overall DSPA application on the Fog. However, FCInterplay adapts the usage of the Fog computational resources according to the data stream rates. In particular, at lower data stream rates (i.e., M1 to M2), FCInterplay has higher Fog computational resource usage than aTSOO-H and TSOO-H as by partially processing each individual data stream S_j on the Fog, the resulting end-to-end latency satisfies the response time constraint (i.e., $L_j \leq Tmax$). Consequently the constraints $T \leq Tmax$ and $B \leq Bmax$ are satisfied. In this way the Fog to Cloud bandwidth resources are saved for other applications. For the next data stream rates (i.e., M3 to M6), we observe that the Fog computational resource usage of FCInterplay starts decreasing when comparing to aTSOO-H (and TSOO-H). In this respect, FCInterplay favors processing entirely the data stream S_j in the Cloud whose when partially processed in the Fog, their resulting end-to-end latency did not satisfy the response time constraint (i.e., $L_j \leq Tmax$). In this way FCInterplay saves the Fog resources for other applications. Finally, at the data stream rate $M7$, we observe that FCInterplay has a sudden increase in the Fog resource usage compared for instance to data stream rate produced by $M6$ IoT devices. This is due not only in the increase of data stream rates, it is also due to the fact that for certain data stream S_j that should be processed entirely in the Cloud, however the constraint $B \leq Bmax$ was not satisfied. Hence, theses data streams are partially processed in the Fog.

We observe that aTSOO-H keeps trying to jointly optimise the usage of the Fog resources and Fog to Cloud network resources.

7.2.2. Fog to Cloud network bandwidth usage ratio

At lowest data stream rates (i.e., M1 and M2), Figure 12b shows that FCInterplay has the lowest Fog to Cloud bandwidth usage ratio thanks to $dataMinCut$ algorithm which was applied in order to partially process data streams S_j in the Fog. For the other data stream rates (i.e., M3 to M7) as they are increasing, the Fog to Cloud bandwidth usage ratio of FCInteraplay is also increasing. It becomes even the highest among all the algorithms at the data stream rate produced by $M6$ and $M7$. This is due to the fact that FCInterplay favors processing some data streams in the Cloud if partially processing them in the Fog would not satisfy the response time constraint.

At the lowest data stream rates (i.e., M1 and M2), FogOnly has the second lowest Fog to Cloud bandwidth usage ratio when comparing to FCInteraplay. However the data stream rates is increasing, FogOnly has the lowest Fog to Cloud bandwidth usage ratio among all the algorithms. FogOnly does not apply $dataMinCut$ as $FCInterplay$. However, given that it replicates as much as possible the operators for each individual data streams on the Fog, as we go from the source to the sink the cumulated selectivity and hence data stream rates most often decrease. As a result the Fog to Cloud bandwidth usage also decreases.

On the other hand, aTSOO-H and TSOO-H have slightly higher usage of these resources when comparing to FogOnly but lower when comparing to FCInterplay. This is due to the fact that aTSOO-H and TSOO-H aim to jointly minimise the Fog resources and Fog to Cloud network resources.

7.2.3. DSPA Application deployment success rate

Figure 12c shows that FCInterplay successfully deploys the DSPA applications only at lower data stream rates (M1 to M2) with success rate of 100%. Upon data stream rates produced by $M3$ the success rate start decreasing. In this respect, FCInterplay has the lowest success rate and it becomes even 0 at data stream rates produced by $M6$ and $M7$. As the data stream rate is increasing the strategy of FCInterplay is not sufficient to solve the TSOO problem, an optimization approach is necessary.

On the other hand FogOnly has the second lowest success rate and it achieves 100% of success rate only at the lowest data stream rates (i.e., M1 and M2). This is due to the choice of maximizing Fog resource usage by replicating as much as possible the operators on the Fog

nodes, as a results operator latencies (processing times) are higher that increase the response time. Hence, the constraint $T \leq Tmax$ is less likely to be satisfied. It worth noting that, the major cause in the decrease of the success rate of FogOnly is the violation of the response time constraint. This is exacerbated by the violation of the Cloud bandwidth constraint at higher data stream rates.

TSOO-H has 100% of success rate from $M1$ to $M4$ while aTSOO-H has 100% of success rate only from $M1$ to $M2$. The success rate of TSOO-H starts decreasing only from $M5$, while aTSOO-H has a success rate lower than TSOO-H but higher than FCInterplay.

8. Conclusion

In this work, we addressed the problem of adaptive scheduling continuous operators of DSPA applications between the Fog and Cloud nodes inline with dynamic data stream rates, as well as, dynamic resource capacities of shareable EFC resources. The objective was to continuously optimise the combined usage cost of the Fog computational resources and the Fog to Cloud network resources while satisfying the response time constraint of a DSPA application. In this respect, we introduced a resource usage cost model that takes into account both maximum and available resource capacities when deploying DSPA applications across the EFC resources and covers that these resources can be shared among several DSPA applications. Then, we introduced the aTSOO-H algorithm that adaptively schedules a DSPA application by taking into account its current deployment state. Using simulations we demonstrate that aTSOO-H efficiently trade-offs the response time, the usage cost of the two types of resources and the execution cost of the scheduling algorithm. The proposed solution in this work for adaptive scheduling a DSPA application encompasses a monitoring tool of resource usage and response time based on witch the aTSOO-H algorithm can be triggered if a problem constraint is not satisfy anymore or if the resource usage goes beyond a threshold value. For evaluating the the proposed algorithm against related work algorithms, we trigger the scheduling algorithms at each change in the data stream rates, therefore as future work, we plan to evaluate aTSOO-H algorithm along with the monitoring tool. In this case, it is necessary to identify the threshold specifically for the overall resource usage cost in order to avoid over or lower threshold estimation. Furthermore, we want to consider not only reactive approach but also predictive approach. the latter enables to trigger the rescheduling of DSPA applications proactively in order to anticipate any degradation in the performances of DSPA applications. Furthermore, we plan to include also computational resources of the Edge layer. In this respect, it will require to extends the system modelling in order to take into account mobility constraint of mobile IoT devices at Edge, as well as, energy consumption.

References

[1] A. Shahid, P. Kang, P. Lama, S. U. Khan, Some new observations on slo-aware edge stream processing, in: 2023 IEEE Cloud Summit, 2023, pp. 27–32. doi:10.1109/CloudSummit57601.2023.00011.

[2] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, K. Tzoumas, Apache flink: Stream and batch processing in a single engine, The Bulletin of the Technical Committee on Data Engineering 38 (4) (2015).

[3] N. Garg, Apache Kafka, Packt Publishing Ltd, 2013.

[4] S. Zeuch, E. T. Zacharatou, S. Zhang, X. Chatziliadis, A. Chaudhary, B. Del Monte, D. Giouroukis, P. M. Grulich, A. Ziehn, V. Mark, Nebulastream: Complex analytics beyond the cloud, Open Journal of Internet Of Things (OJIOT) 6 (1) (2020) 66–81.

[5] H. Röger, R. Mayer, A comprehensive survey on parallelization and elasticity in stream processing, ACM CSUR (2019).

[6] A. Jonathan, A. Chandra, J. Weissman, Wasp: Wide-area adaptive stream processing, in: Proceedings of the 21st International Middleware Conference, 2020, pp. 221–235.

[7] H. K. Apat, R. Nayak, B. Sahoo, A comprehensive review on internet of things application placement in fog computing environment, Internet of Things (2023) 100866.

[8] M. Rzepka, P. Boryło, M. D. Assunção, A. Lasoń, L. Lefèvre, Sdn-based fog and cloud interplay for stream processing, Future Generation Computer Systems (2022).

[9] A. Ali-Eldin, B. Wang, P. Shenoy, The hidden cost of the edge: A performance comparison of edge and cloud latencies, arXiv preprint arXiv:2104.14050 (2021).

[10] P. Ntumba, N. Georgantas, V. Christophides, Scheduling of continuous operators for iot edge analytics with time constraints, in: SMARTCOMP 2022-International Conference on Smart Computing, 2022.

[11] J. Traub, S. Breß, T. Rabl, A. Katsifodimos, V. Markl, Optimized on-demand data streaming from sensor nodes, in: Proceedings of the 2017 Symposium on Cloud Computing, 2017, pp. 586–597.

[12] U. Srivastava, K. Munagala, J. Widom, Operator placement for in-network stream query processing, in: Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, 2005, pp. 250–258.

[13] F. R. De Souza, Scheduling solutions for data stream processing applications on cloud-edge infrastructure, Ph.D. thesis, Université de Lyon (2020).

[14] A. da Silva Veith, M. D. de Assuncao, L. Lefevre, Latency-aware strategies for deploying data stream processing applications on large cloud-edge infrastructure, IEEE transactions on cloud computing (2021).

[15] A. da Silva Veith, Quality of service aware mechanisms for (re) configuring data stream processing applications on highly distributed infrastructure, Ph.D. thesis, Université de Lyon (2019).

[16] A. da Silva Veith, M. D. de Assunçao, L. Lefevre, Monte-carlo tree search and reinforcement learning for reconfiguring data stream processing on edge computing, in: 2019 31st International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), IEEE, 2019, pp. 48–55.

[17] P. Neophytou, J. Szwedko, M. A. Sharaf, P. K. Chrysanthis, A. Labrinidis, Optimizing the energy consumption of continuous query processing with mobile clients, in: 2011 IEEE 12th International Conference on Mobile Data Management, Vol. 1, IEEE, 2011, pp. 98–103.

[18] P. Neophytou, M. A. Sharaf, P. K. Chrysanthis, A. Labrinidis, Power-aware operator placement and broadcasting of continuous query results, in: Proceedings of the Ninth ACM International Workshop on Data Engineering for Wireless and Mobile Access, 2010, pp. 49–56.

[19] L. Prosperi, A. Costan, P. Silva, G. Antoniu, Planner: cost-efficient execution plans placement for uniform stream analytics on edge and cloud, in: 2018 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS), IEEE, 2018, pp. 42–51.

[20] L. Yang, J. Cao, Y. Yuan, T. Li, A. Han, A. Chan, A framework for partitioning and execution of data stream applications in mobile cloud computing, ACM SIGMETRICS Performance Evaluation Review 40 (4) (2013) 23–32.

[21] F. R. d. Souza, A. D. Silva Veith, M. Dias de Assunção, E. Caron, Scalable joint optimization of placement and parallelism of data stream processing applications on cloud-edge infrastructure, in: International Conference on Service-Oriented Computing, Springer, 2020, pp. 149–164.

[22] F. R. de Souza, M. D. de Assunçao, E. Caron, A. da Silva Veith,

An optimal model for optimizing the placement and parallelism of data stream processing applications on cloud-edge computing, in: 2020 IEEE 32nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), IEEE, 2020, pp. 59–66.
[23] G. Amarasinghe, M. D. De Assuncao, A. Harwood, S. Karunasekera, A data stream processing optimisation framework for edge computing applications, in: 2018 IEEE 21st International Symposium on Real-Time Distributed Computing (ISORC), IEEE, 2018, pp. 91–98.
[24] H. P. Sajjad, K. Danniswara, A. Al-Shishtawy, V. Vlassov, Spanedge: Towards unifying stream processing over central and near-the-edge data centers, in: 2016 IEEE/ACM Symposium on Edge Computing (SEC), IEEE, 2016, pp. 168–178.
[25] L. Li, M. Guo, et al., Online workload allocation via fog-fog-cloud cooperation to reduce iot task service delay, Sensors (2019).
[26] M. Peixoto, T. Genez, L. F. Bittencourt, Hierarchical scheduling mechanisms in multi-level fog computing, IEEE Transactions on Services Computing (2021).
[27] F. R. de Souza, M. D. de Assunçao, E. Caron, A throughput model for data stream processing on fog computing, in: 2019 International Conference on High Performance Computing & Simulation (HPCS), IEEE, 2019, pp. 969–975.
[28] M. Nardelli, Qos-aware deployment and adaptation of data stream processing applications in geo-distributed environments, Ph.D. thesis, Ph. D. thesis, University of Rome Tor Vergata (2018).
[29] V. Cardellini, V. Grassi, F. Lo Presti, M. Nardelli, Optimal operator placement for distributed stream processing applications, in: Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems, 2016, pp. 69–80.
[30] M. Nardelli, V. Cardellini, V. Grassi, F. L. Presti, Efficient operator placement for distributed data stream processing applications, IEEE TPDS (2019).
[31] H. Arkian, G. Pierre, J. Tordsson, E. Elmroth, Model-based stream processing auto-scaling in geo-distributed environments, in: ICCCN, 2021.
[32] V. Issarny, B. Billet, G. Bouloukakis, D. Florescu, C. Toma, Lattice: A framework for optimizing iot system configurations at the edge, in: 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS), IEEE, 2019, pp. 1797–1805.
[33] P. Varshney, Y. Simmhan, Demystifying fog computing: Characterizing architectures, applications and abstractions, in: 2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC), IEEE, 2017, pp. 115–124.
[34] R. Prasad, C. Dovrolis, M. Murray, K. Claffy, Bandwidth estimation: metrics, measurement techniques, and tools, IEEE network 17 (6) (2003) 27–35.
[35] T. Djemai, P. Stolf, T. Monteil, 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, pp. 1–7.
[36] J. Edmonds, R. M. Karp, Theoretical improvements in algorithmic efficiency for network flow problems, JACM (1972).
[37] P. Ntumba, N. Georgantas, V. Christophides, Efficient scheduling of streaming operators for iot edge analytics, in: Accepted to the 6th International Conference on Fog and Mobile Edge Computing, 2021.
[38] P. Ntumba, N. Georgantas, V. Christophides, Scheduling continuous operators for iot edge analytics, in: Proceedings of the 4th International Workshop on Edge Systems, Analytics and Networking, 2021, pp. 55–60.
[39] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, M. Seltzer, Network-aware operator placement for stream-processing systems, in: 22nd International Conference on Data Engineering (ICDE'06), IEEE, 2006, pp. 49–49.
[40] S. Rizou, F. Diirr, K. Rothermel, Fulfilling end-to-end latency constraints in large-scale streaming environments, in: IPCCC, 2011.
[41] F. Dabek, R. Cox, F. Kaashoek, R. Morris, Vivaldi: A decentralized network coordinate system, ACM SIGCOMM Computer Communication Review 34 (4) (2004) 15–26.
[42] 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.
[43] J. F. Shortle, J. M. Thompson, D. Gross, C. M. Harris, Fundamentals of queueing theory, Vol. 399, John Wiley & Sons, 2018.
[44] K. Patroumpas, T. Sellis, Window specification over data streams, in: International Conference on Extending Database Technology, Springer, 2006, pp. 445–464.
[45] Q. Jiang, S. Chakravarthy, Queueing analysis of relational operators for continuous data streams, in: Proceedings of the twelfth international conference on Information and knowledge management, 2003.
[46] B. Jansson, Choosing a good appointment system—a study of queues of the type (d, m, 1), Operations Research 14 (2) (1966) 292–312.
[47] Amazon, Instances m5 amazon ec2, `https://aws.amazon.com/fr/ec2/instance-types/m5/`, [Online; accessed 30-August-2022] (2021).
[48] V. A. Strusevich, Shop scheduling problems under precedence constraints, Annals of operations research 69 (1997).
[49] Y.-W. Hung, Y.-C. Chen, C. Lo, A. G. So, S.-C. Chang, Dynamic workload allocation for edge computing, VLSI (2021).
[50] T. Rausch, C. Lachner, P. A. Frangoudis, P. Raith, S. Dustdar, Synthesizing plausible infrastructure configurations for evaluating edge computing systems, in: 3rd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 20), 2020.
[51] C. Wöbker, A. Seitz, H. Mueller, B. Bruegge, Fogernetes: Deployment and management of fog computing applications, in: Network Operations and Management Symposium, IEEE, 2018, pp. 1–7.
[52] 7-Zip LZMA Benchmark.
URL `https://www.7-cpu.com/`
[53] P. Silva, A. Costan, et al., Investigating edge vs. cloud computing trade-offs for stream processing, in: IEEE Big Data, 2019.